Computer Graphics

Andreas Kriegl

July 23, 2003

These are the notes for the lecture course with the same title which I gave in the Summer-Semester 2003. It contains all the material I treated and offers also some additional informations. I inserted lots of links to the www but, of course, can not guarantee, that they will persist. I also included links to the original documentation of **Pov-Ray**. They are denoted like [pov:1.1.1] . In some chapters links are collected at the beginning, in others they are spread throughout the text. I prepended them sometimes with ★'s in order to indicate their subjective relevance for this lecture course.

I would be very happy to receive any feedback for these notes and I will try to take it into account in the next version.

Remains to wish you all an inspiring reading,

Andreas Kriegl

Contents

Chapter 1

Color

Links

★★★ William Shoaff: Color, Illumination Models, and Shading
www.cs.fit.edu/. . . /index.html
www.cs.fit.edu/. . . /illuminate.pdf
keywords: lecture on light, illumination, shading

★★ Heckbert-15462: light_2.pdf
almond.srv.cs.cmu.edu/. . . /light_2.pdf
keywords: slides on light, color, illumination

★ Heckbert-15462: illum.pdf
almond.srv.cs.cmu.edu/. . . /illum.pdf
keywords: slides on light, color, reflection, shadows, texture

★ Wikipedia: Color - Wikipedia
Colour.html
keywords: rainbow, color vision, primary colors, links!

★ :Graphics
graphics.html
keywords: pictures!, cietoppm!, rainbow-prism, links

★★★ Hyper Physics: CIE Color System
cie-1.html
keywords: CIE, Color regions, links; CIE diagrams

★★ Adobe: The CIE Color Models - Technical Guides
hyperphysics.phy-astr.gsu.edu/. . . /cie.html
keywords: CIE models, Observer, links!

★ Adobe: CIEXYZ - Color Models - Technical Guides
www.adobe.com/. . . /ciexyz.html
keywords: CIE xyz-model

★★★ Adobe: CIELUV - Color Models - Technical Guides
www.adobe.com/. . . /cieluv.html
keywords: CIE LUV model, color distances

★ Gordon W. Braudaway, Hon-Sum P. Wong: Color science
colorsci.html
keywords: CIE 1931, Grassmann's law!

★★★ Hector Xiang: efg's Chromaticity Diagrams Lab Report
Chromaticity.htm
keywords: CIE 1931, color matching functions, chromaticity diagrams and coordinates

★★ R.Nave, Hyper Physics: The 1976 Revision of the CIE Color System
hyperphysics.phy-astr.gsu.edu/. . . /cie1976.html
keywords: CIE 1976

★★★ Color and Vision Research Labs: CVRL Color & Vision database
CVRL Color & Vision database_files/index.html
keywords: links to data files, CIE

★ Color and Vision Research Labs: CVRL Color & Vision database
CVRL Color & Vision database.html
keywords: CIE in xyz-coordinates

★ :sbrgb2.txt
cvrl.ioo.ucl.ac.uk/. . . /sbrgb2.txt
keywords: data

★ :ssmb_1.txt
cvrl.ioo.ucl.ac.uk/. . . /ssmb_1.txt
keywords: data

★ :ciexyz31_1.txt
cvrl.ioo.ucl.ac.uk/. . . /ciexyz31_1.txt
keywords: data, CIE xyz-functions; table

★ :ciexyz31.txt
cvrl.ioo.ucl.ac.uk/. . . /ciexyz31.txt
keywords: data, CIE xyz-functions; table

★ Wikipedia: Blackbody - Wikipedia
www.wikipedia.org/. . . /Black_body
keywords: Blackbody, Planck's Law, links; Planck's Law of Radiation

★★ :Radiation Laws
csep10.phys.utk.edu/. . . /radiation.html
keywords: Plank's radiation law, jawa, links

★ :Color Indices and Surface Temperature
csep10.phys.utk.edu/. . . /cindex.html
keywords: color temperature, links, astronomy

★★ Adobe: Light and Color - Basic Color Theory for the Desktop - Technical Guides
www.adobe.com/. . . /light.html
keywords: illuminants, color temperature, power distribution of light sources

★ Cybaea: Color Temperature and Color Correction in Photography
cybaea.com/. . . /color-correction.html
keywords: color temperature, photography, filters

★★ M.Abramowitz, T.J.Fellers, M.W.Davidson: Molecular Expressions Microscopy Primer: Light and Color - Color Temperature
micro.magnet.fsu.edu/. . . /colortemperatureintro.html
keywords: color temperature, java, links, photography, filters

★ Steve: Color Temperature and Visual Color Perception
colortemp.html
keywords: color temperature, astronomy

★ :plabpc.csustan.edu.html
plabpc.csustan.edu/. . . /index.html
keywords: color temperature, astronomy

★ All Griffin: SoundVisionBehindNumbers
SoundVisionBehindNumbers isf article.pdf
keywords: color temperature, CIE

★★ Tom Flynn: The Color of "White"; Beyond Belief (Skeptical Briefs September 1994)
www.csicop.org/. . . / t+beyond+belief
keywords: color temperature, visual processing

★ Mike Rollins: The Color Temperature of Light
ColorTemp.htm
keywords: color temperature, monitors

★★ Alex Byne, David Hilbet: Glossary of Color Science
tigger.uic.edu/. . . /Glossary.html
keywords: glossary on color

Color is treated in many subjects: physics, physiology, psychology, art, graphic design and astronomy, The colors we see depend at least on the light source, the objects we view, their surroundings, the human visual system. To make things a little simpler in the beginning we start with achromatic light.

1.1 Achromatic Light

This is colorless light described by the notions black–gray–white. Its main quality is INTENSITY or LUMINANCE, which is the quantity of light in the physics sense of energy. In contrast BRIGHTNESS is the perceived intensity in the psychological sense.

1.1.1 Gamma Correction

What is the relationship between intensity and brightness? Let us scale brightness and intensity in such a way that 1.0 is assigned to the maximal value. For a monitor it is clear what is meant by maximal value; on the other hand there is a minimal intensity we denote I_0. Note that $I_0 > 0$, i.e. absolute black is not possible, since the phosphors in the CRT (Cathode Ray Tube) reflect light. The proportion $1/I_0$ is called the DYNAMICAL RANGE of the monitor and is for CRTs usually typically 50 and 200.

Since on the computer we can only code a finite number of levels inbetween I_0 and 1, we have to decide for which intensity levels they should stand for. Linear spacing like

$$I_0, I_0 + \Delta, I_0 + 2\Delta, \ldots, I_0 + n\Delta = 1.0$$

for some increment $\Delta > 0$ is not a good idea, since the eye is sensitive to ratios of intensity levels rather than absolute values of intensity. Thus we should space intensity levels logarithmically, i.e.

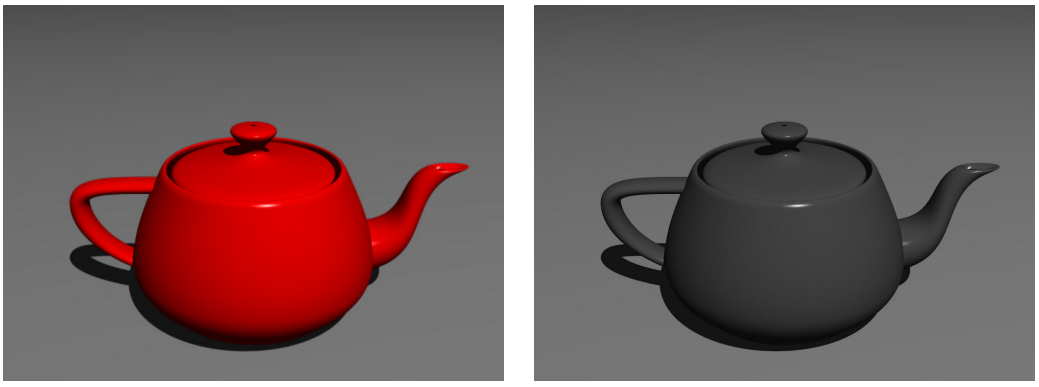
$$I_0, r\,I_0, r^2\,I_0, \ldots, r^n\,I_0 = 1.0$$

for some factor $r > 1$. For a dynamical range of say $100 = 1/I_0$ and $n = 256$ steps (as they can be coded by one byte in the range 0...FF) we obtain the factor $r = \sqrt[n]{1/I_0} \approx 1.01815$ corresponding to a 1.8% intensity increase for each step. The approximate intensity increase the human eye can detect is about 1.0%, i.e. $r = 1.01$. So we need $n \geq \log_r(1/I_0) \approx 463$ many steps to display a continuous scale from I_0 to 1.0 on a display with a dynamical range of $1/I_0 = 100$. Consequently, coding gray values using one byte(=8 bits) per pixel is not enough to ensure a continuous gray scale. In practice for B&W printing slight blurring due to ink bleeding and random noise reduces the number of distinguishable intensities to approximately 64.

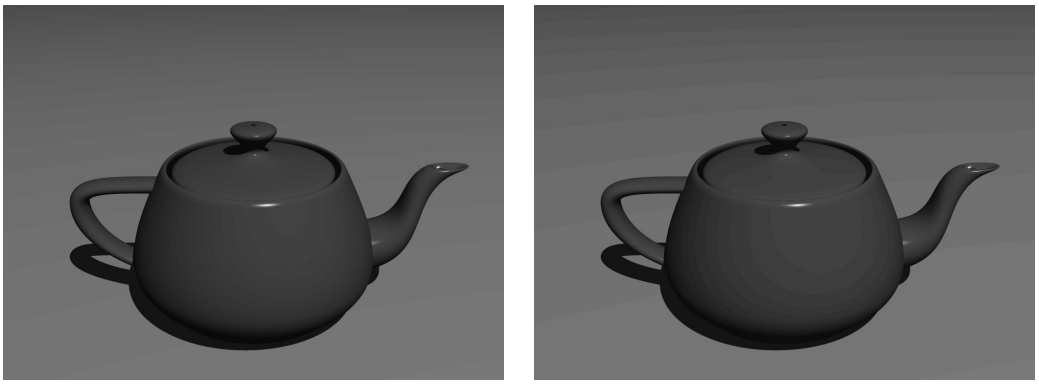
Display media	Dynamic range $1/L_0$	Intensity levels $n = \log_{1.01}(1/L_0)$
CRT	50–200	393–532
Photographic prints	100	463
Photographic slides	1000	694

128 steps

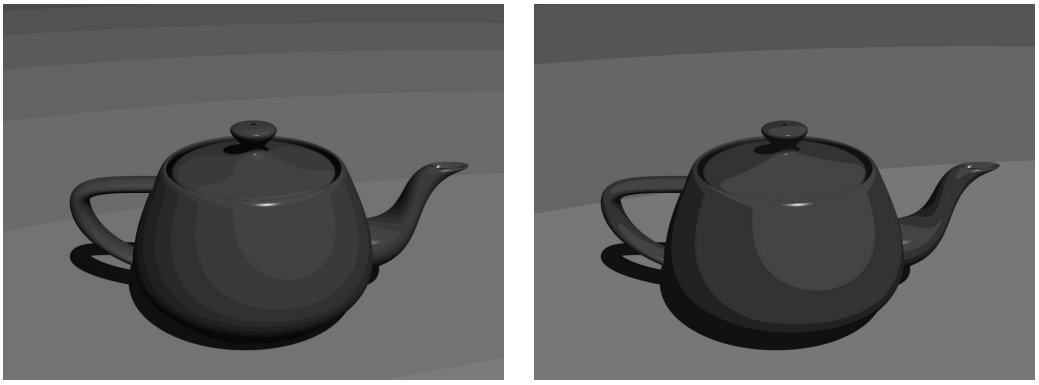
True color



True color & 256 colors



128 & 64 colors



32 & 16 colors

8 & 4 colors

How can these logarithmically spaced intensity levels be displayed? According to the formulas above the i-th level should have intensity

$$I_i = r^i \, I_0.$$

The intensity I of a CRT is proportional to N^γ, where N is the number of electrons and γ is some constant depending on the phosphors used in the CRT and is usually

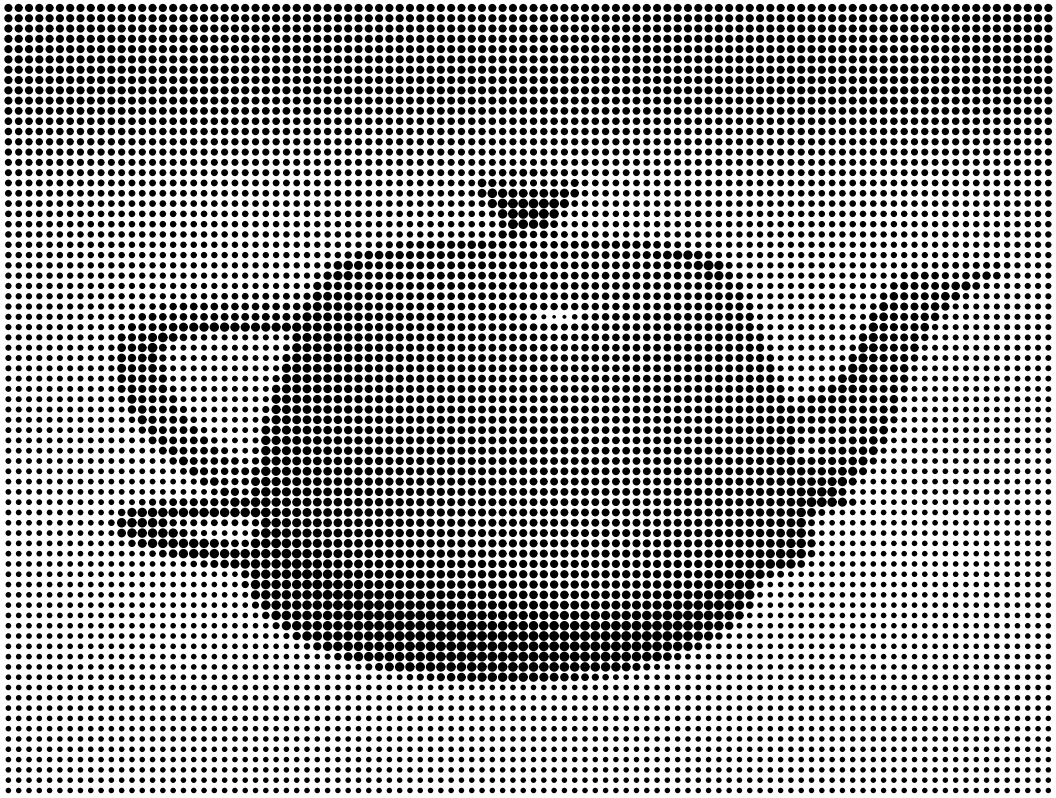
in the range $2.2 \leq \gamma \leq 2.5$. Since the number of electrons emitted is approximately proportional to the applied control-grid voltage V we have

$$I = K\,V^{\gamma} \text{ for some constant } K.$$

So suppose the intensity we want to display is I. The corresponding index i has to be chosen such that I_i/I is as close to 1 as possible, i.e. $i = \text{round}(\log_r(I/I_0))$. The corresponding V is thus $V_i = \sqrt[\gamma]{I_i/K}$. If this conversion is not hardcoded into the display, the software has to take account on this. This is called GAMMA-CORRECTION. Without gamma-correction quantization errors (produced by approximating true intensity levels by discrete displayable ones) are more conspicuous near black than near white.

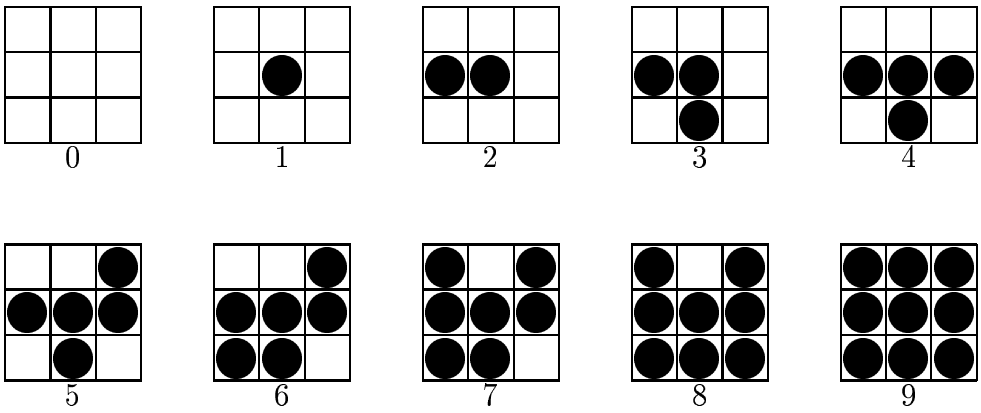
1.1.2 Halftone Approximation

If we cannot display all the required intensity levels (e.g. on a printer) we need a trick using the spatial integration that our eyes performs: In normal light the eye can only detect about one arc minute (1/60 degree). This is called VISUAL ACUITY. Thus instead of gray dots a small black disk with radius varying according to the blackness $1-I$ is printed. Usually, for newspaper 60–80 and for magazines 150–200 different radiuses are used. This process is called HALFTONING.

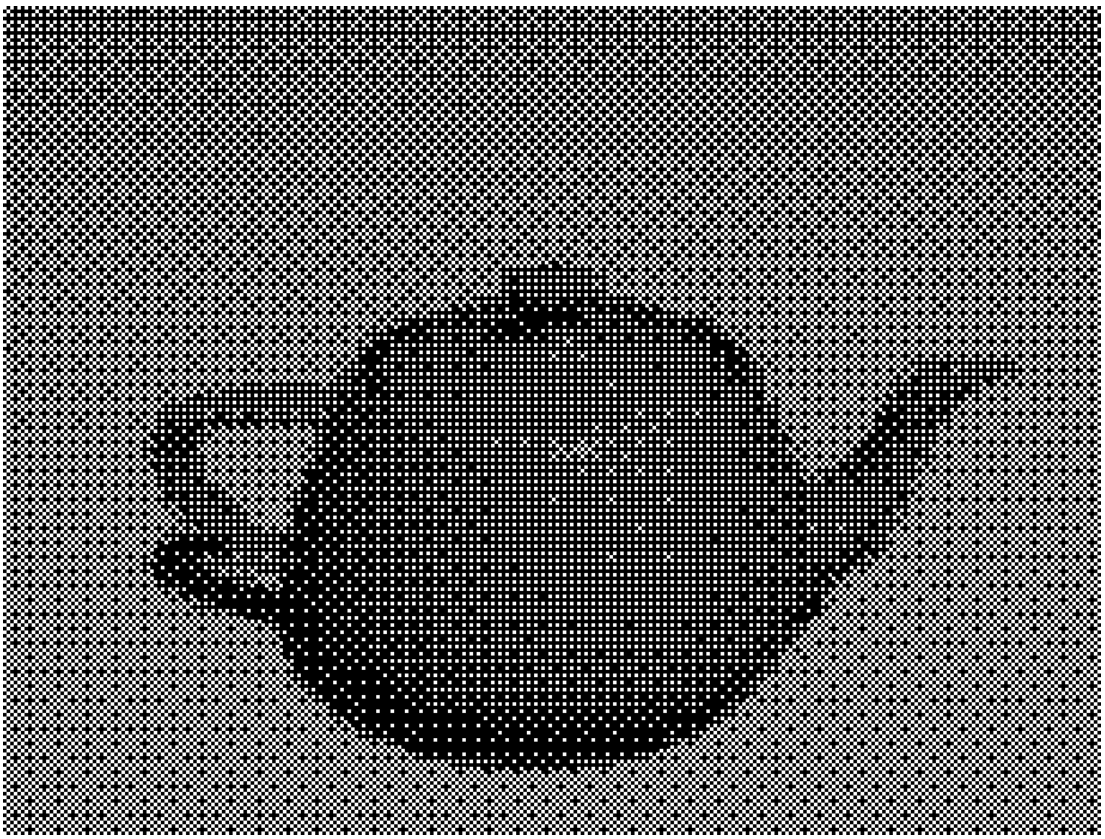


Halftoning

For computers this is implemented as CLUSTERED-DOT ORDERED DITHERING, e.g. the following patterns are used for each pixel of intensity $0 \ldots 9$:



Clustered-dot ordered dithering



Clustered-dot ordered dithering

This can be described by a dither matrix

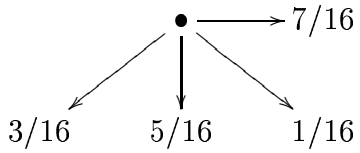
$$\begin{pmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{pmatrix}$$

saying that in order to display intensity I one should turn on all those pixels whose value in this matrix is less than I. It is chosen in such a way, that visual artifacts are avoided:

- Growth sequence to minimize contour effects, i.e. the pattern are subsets of each other (hence the name ordered). They start in the center and expand towards the boundary.
- Keep the one's adjacent to each other (hence the name clustered dot), but this is not necessary for monitors.

For high-quality reproduction we need an 8×8 (64 gray levels) or even a 10×10 matrix and thus quite a hight resolution.

If such a high resolution is not available one can use ERROR DIFFUSION: There we use fewer intensity levels than desired, but we distribute the errors $I - I_i$ to the neighboring pixels with the following weights:



8 colors with Floyd-Steinberg error diffusion

A similar trick can be used for enlarging a picture by taking interpolation values to neighboring pixels as intermediate values. E.g. for doubling the size of the picture one inserts new rows and columns and takes as new values

$$I'_{2i,2j} := I_{i,j} \qquad I'_{2i+1,2j} := \frac{1}{2}(I_{i,j} + I_{i+1,j})$$

$$I'_{2i,2j+1} := \frac{1}{2}(I_{2i,j} + I_{2i,j+1}) \qquad I'_{2i+1,2j+1} := \frac{1}{4}(I_{i,j} + I_{i,j+1} + I_{i+1,j} + I_{i+1,j+1})$$

This way one avoids that small square-size pixels can be seen. Disadvantage is that some blurring occurs.

Enlarged versus enlarged with diffusion

1.2 Chromatic Light

How to describe color? – We have several possibilities:

1. Make a table of colors, e.g. Munsell color-order system.

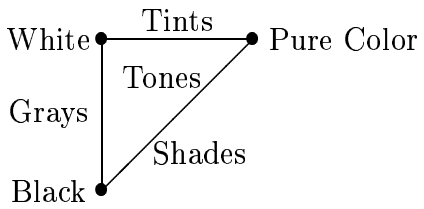
A commercial color table

2. Assign names to colors, e.g. Light-Goldenrot-Yellow, Medium-Spring-Green, etc.

3. Produce colors by some process: Artists speak about the following variations of pure pigments:

- TINTS (means adding white),

- SHADES (means adding black),
- and TONES (means adding a combination of both).



The artists description of colors

Consequently colored light has following quantities associated:

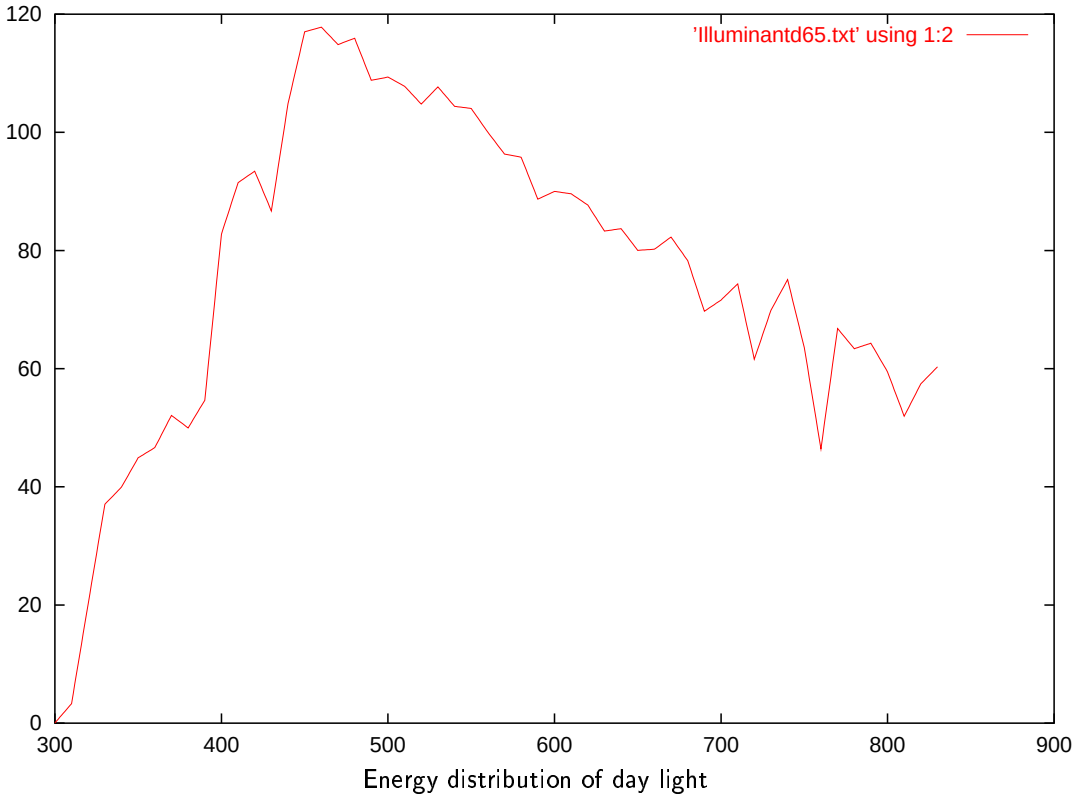
- HUE (corresponding to pure pigment)
- SATURATION (i.e. distance from gray of equal intensity)
- LIGHTNESS (i.e. perceived intensity of a reflecting object)
  or BRIGHTNESS (i.e. perceived intensity of light emitting object).

1.2.1 Psychophysics

Colorimetry:

Light is ELECTROMAGNETIC ENERGY in the 400 nm to 700 nm wavelength, where one nm (nanometer) is $1/10^{-6}$mm. Pure or MONOCHROMATIC LIGHT is perceived as one color in the range violet–indigo–blue–green–yellow–orange–red of the rainbow.

The amount of energy present at each wavelength is represented by the SPECTRAL-ENERGY-DISTRIBUTION of the light. For example, the spectral-energy-distributions of sunlight (what we call white light) is:



Many different distributions produce the same color impression (and are called METAMERS). Thus one can quantify light by:

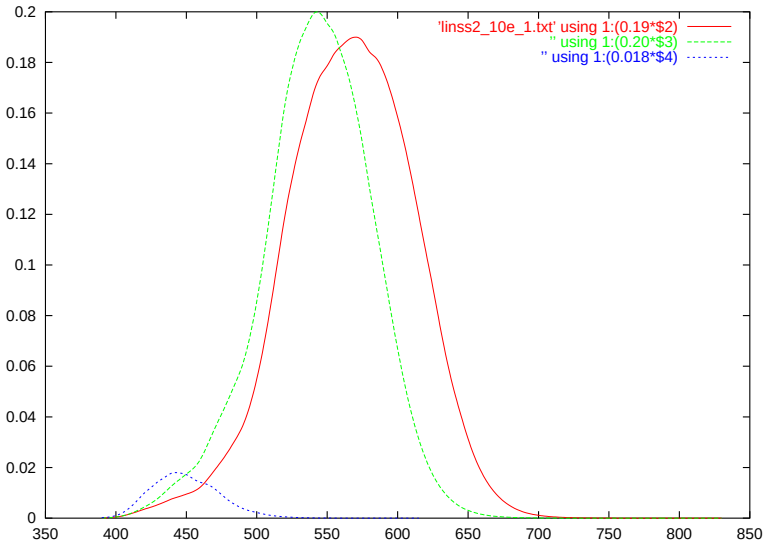
- DOMINANT WAVELENGTH (color we see), which corresponds to hue,
- EXCITATION PURITY (proportion of pure light of dominant wavelength to white light), which corresponds to saturation, and
- LUMINANCE which corresponds to intensity.

Note however, that the dominant wavelength is not necessarily that with largest spectral component.

How is color determined by the human eye? The retina seems to have 3 kind of color sensors (called CONES) with approximative peak sensitivity at red, green and blue. This is the TRI-STIMULUS THEORY of color perception.

The cones and rods in the human retina
From:www.cs.fit.edu/. . . /index.html

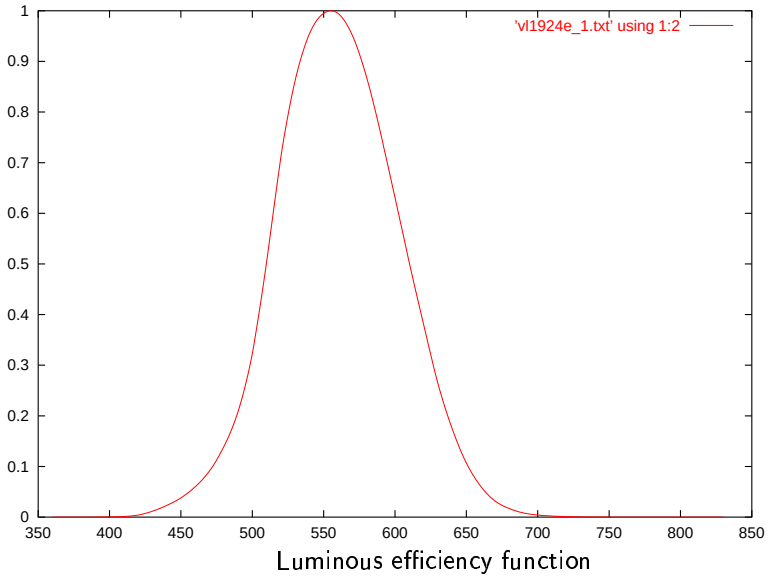
The spectral response function R, G and B of this cones to monochromatic light is following:



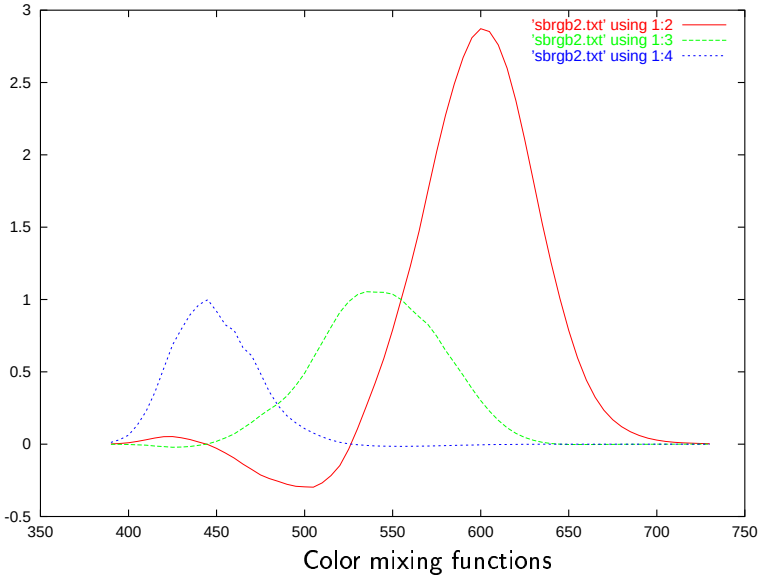
Spectral responses of the cones to monochromatic light at given wavelength

Response to blue light is less strong. Note that the sensitivity peaks of R and G are in the yellow range.

LUMINOUS-EFFICIENCY function (the eye's response to light of constant luminance at given wavelength) corresponds to the sum of the curves above (compare this with the energy distribution of sunlight):



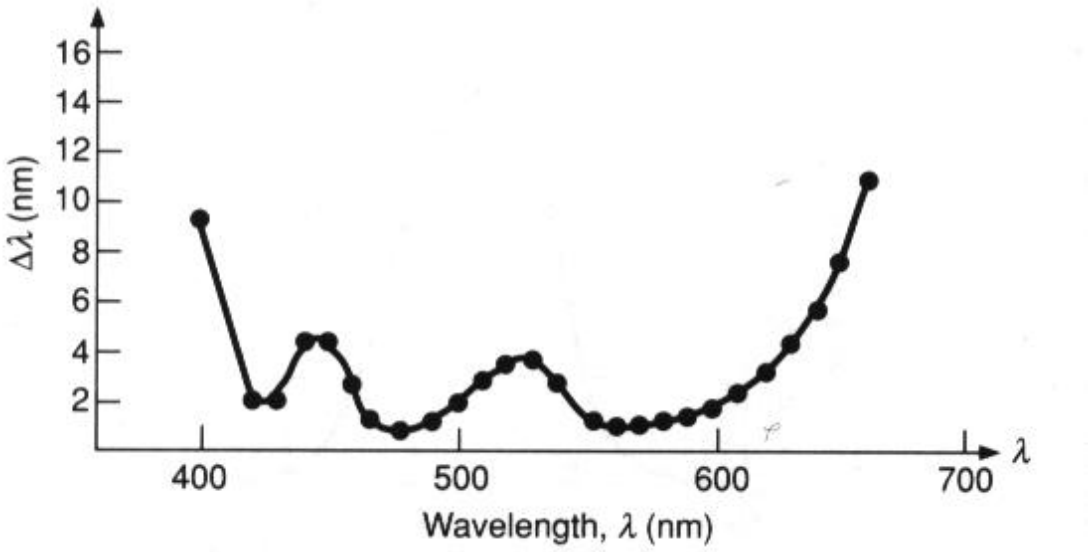
So the idea is, that every by the human eye distinguishable color can be produced by a additive mixture of red, green and blue. And this is the principle of color CRTs. The amount of R-G-B needed to match in the observers eye a monochromatic color of constant luminance has been experimentally determined as:



Note, that negative values of red are necessary in the range from 450nm to 525nm (which means that this amount of red has to be added to the given color in order that this new color can be matched by the described values of G and B). A

consequence is, that certain colors CAN NOT be produced on a CRT by R-G-B-mixes.

The human eye can distinguish hundreds of thousands of colors when shown side by side, see:

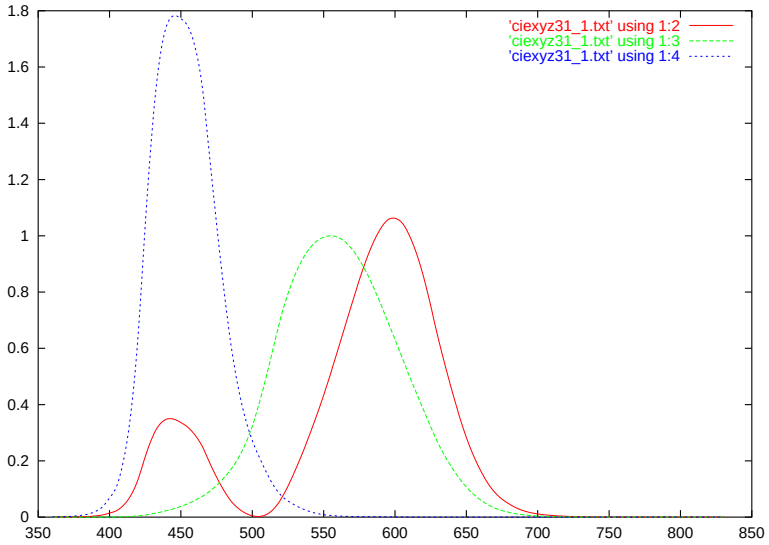


Just-noticeable color differences as a function of the wave-length

This sensitivity to color differences is dependent on the wave length. Approximately 128 fully saturated hues can be distinguished. The eye is less sensitive to hue-changes in less saturated light. Its sensitivity to changes in saturation is greater at the extreme end of spectrum (where there are approximately 23 steps). In contrast, at 575 nm only 16 saturations steps can be distinguished.

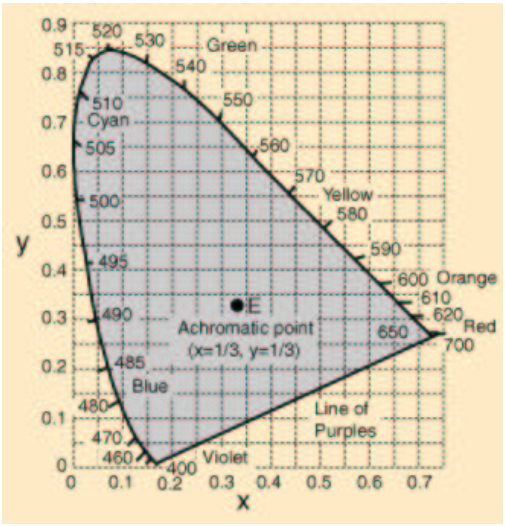
1.2.2 The CIE Chromaticity Diagram

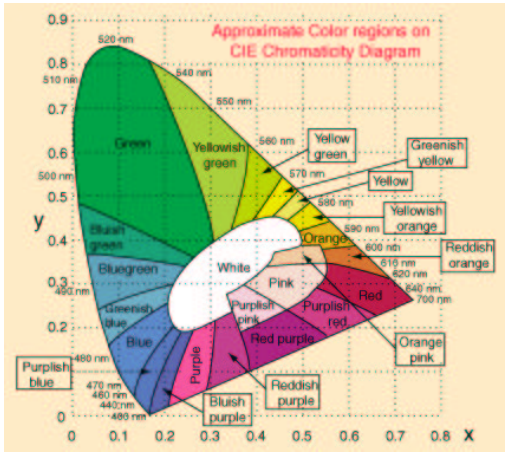
The negative values in the representation of color by R-G-B-values is unpleasant. Thus the Commission Internationale de l'Éclairage (CIE) defined in 1931 another base in terms of (virtual) primaries $\mathbf{X}$, (the luminous-efficiency function) $\mathbf{Y}$ and $\mathbf{Z}$, which allows to match all visible colors as linear combinations with positive coefficients only (the so called CHROMATICITY VALUES X, Y, Z), i.e. any visible color $\mathbf{C}$ can be expressed as $\mathbf{C} = X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z}$, see



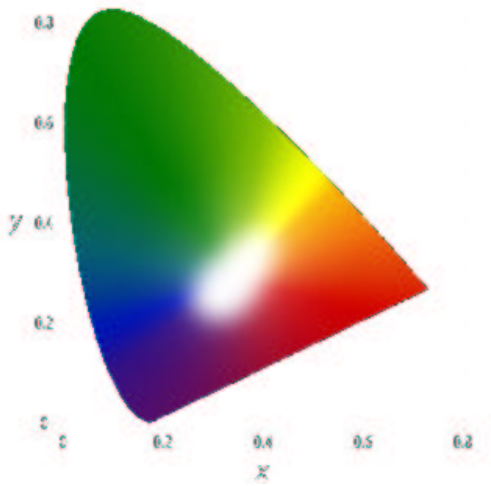
CIE 1931 primaries

Normalization to $X + Y + Z = 1$ gives new coordinates x, y (and $z = 1 - x - y$), which are independent on luminous energy $X+Y+Z$. The visible chromatic values in this coordinate system form a horseshoe shaped region, with the spectrally pure colors on the curved boundary. Warning: brown is orange-red at very low luminance (hence is not shown in this diagram). Standard white light (approximative sunlight) is located at point C near $x = y = z = 1/3$.





CIE 1931 Chromaticity Diagram
From:cie-1.html



Horseshoe of visible colors
From:www.adobe.com/. . . /ciexyz.html

The DOMINANT WAVELENGTH of some color is given by the intersection of the ray from C to the color with the curved boundary formed by the pure colors.

Some colors (purples and magentas) are non-spectral, i.e. have no dominant wavelength (since the intersection of the rays hit the boundary in the flat part). But they have have a COMPLEMENTARY DOMINANT WAVELENGTH, lying on the opposite side.

COLOR COMPLEMENTARY to some color are opposite to C on the line through C. E.g. we have the following complementary pairs: red–cyan, green–magenta, and blue–yellow.

EXCITATION PURITY is a ratio of the distances from the color and the dominant wavelength to C.

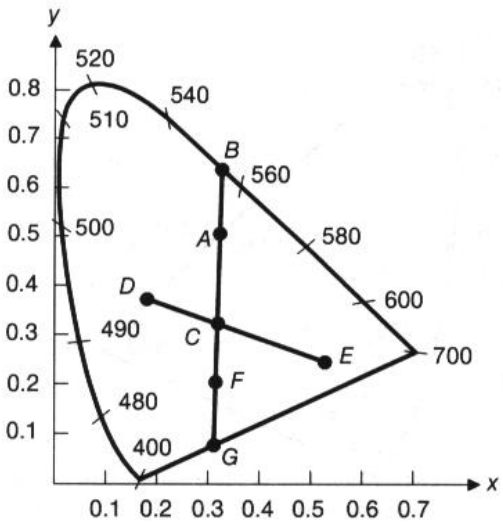
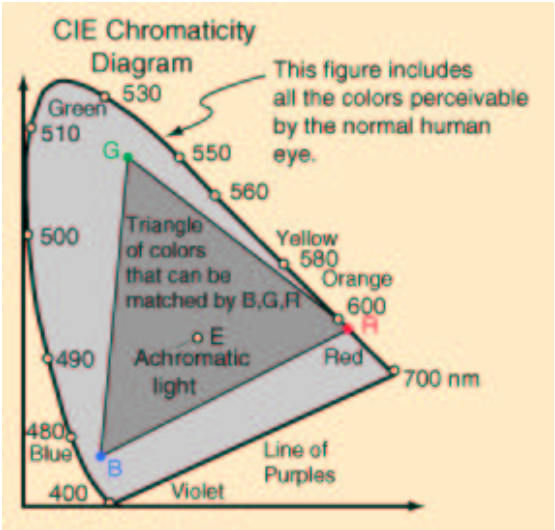


Fig. 13.25 Colors on the chromaticity diagram. The dominant wavelength of color *A* is that of color *B*. Colors *D* and *E* are complementary colors. The dominant wavelength of color *F* is defined as the complement of the dominant wavelength of color *A*.

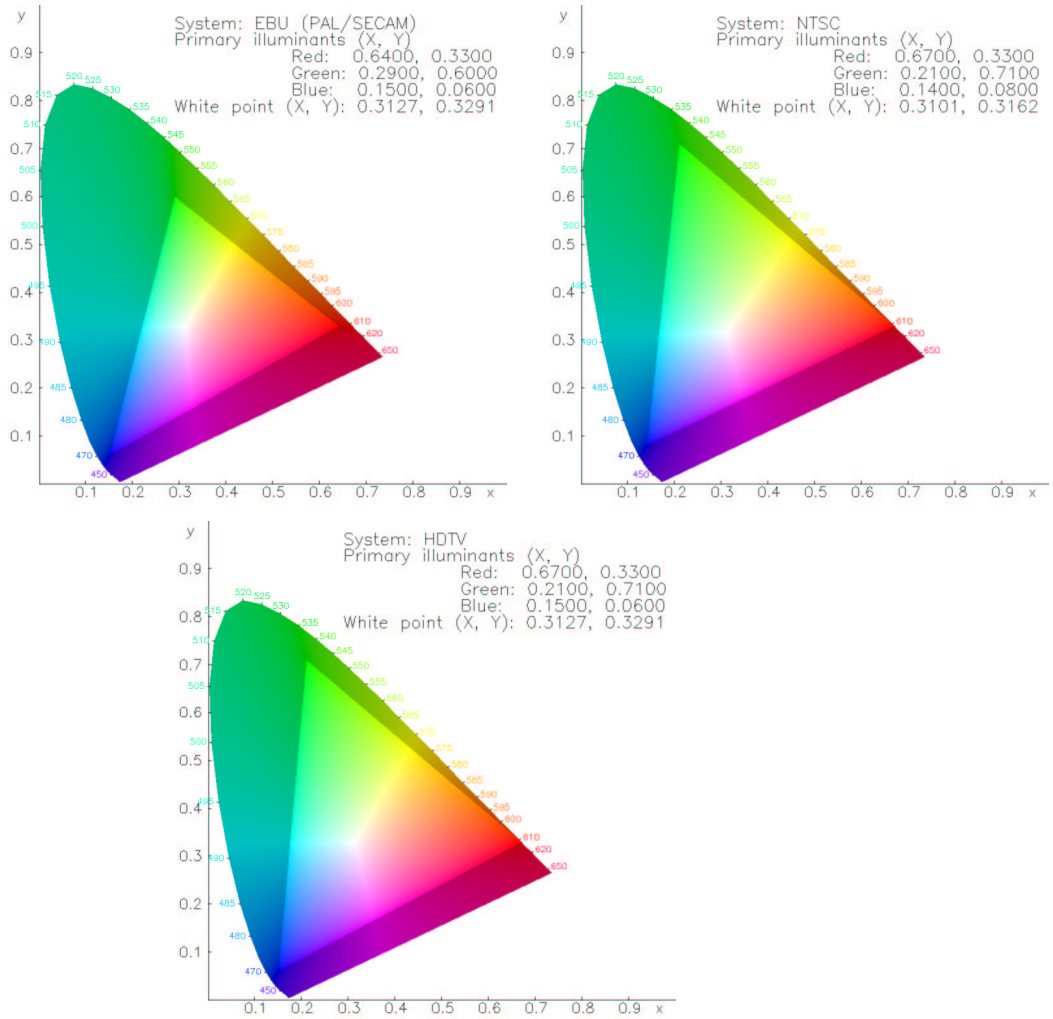
Dominant wavelength and complementary colors

The CIE chromaticity diagram can also be used to visualize the COLOR GAMUTS (i.e. the ranges of producible colors) for various output devices:



Color gamut

From:cie-1.html



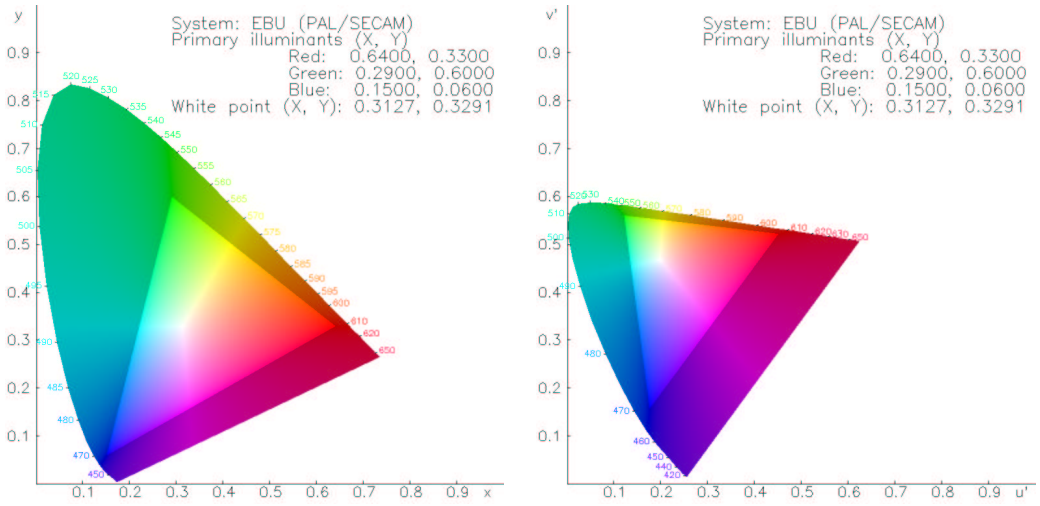
Color gamuts for PAL, NTSC and HDTV and their chromaticity values

The chromaticity values for standard NTSC RGB phosphor are:

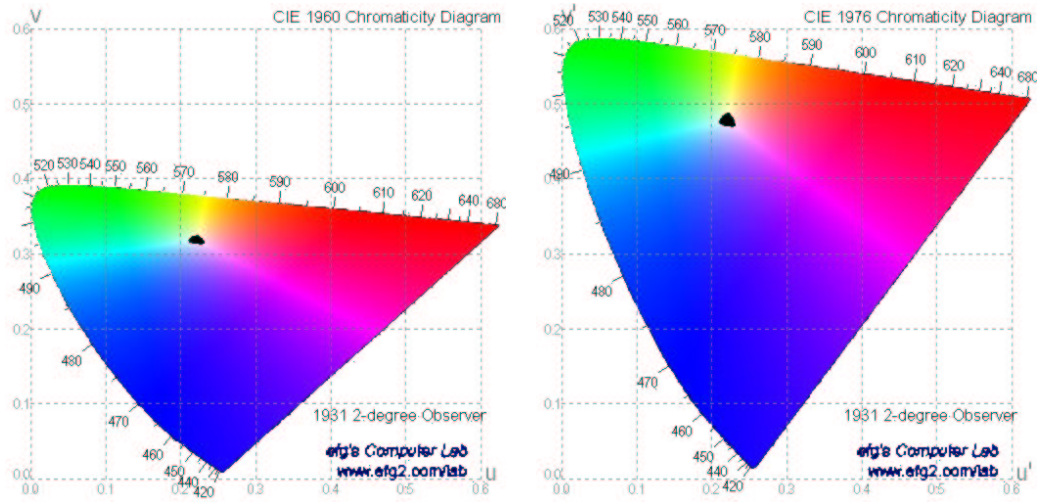
	R	G	B
x	0.67	0.21	0.14
y	0.33	0.71	0.08

The color-printer gamut is rather small in comparison to the color-monitor gamut. Thus we can print much fewer colors that we can display on the screen.

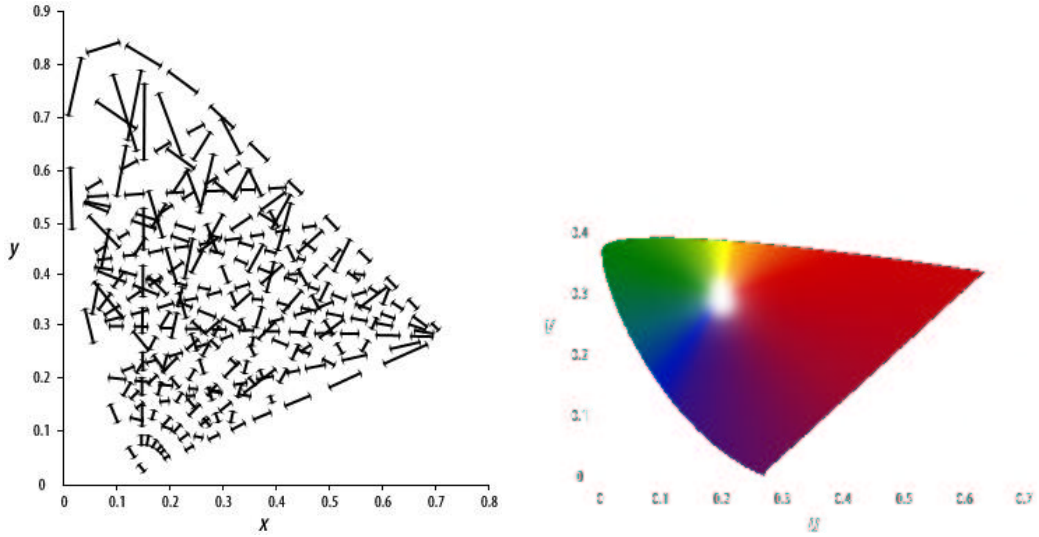
A disadvantage of the CIE 1931 standard is that equal distances in the $X - Y$ coordinates are not perceived as being equal. This was corrected by 1976 CIE LUV standard.



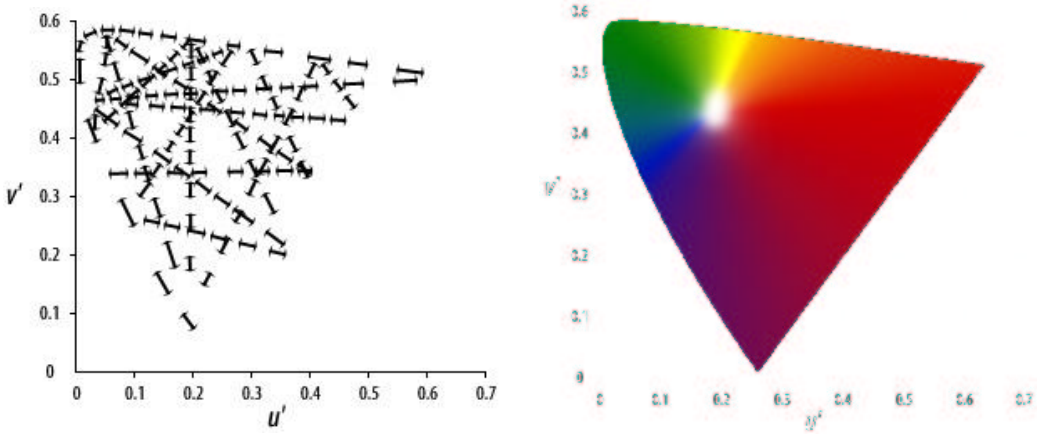
Chromaticity diagrams of 1931 and of 1976



Chromaticity diagrams of 1960 and 1976
From:Chromaticity.htm



Equally perceived color distances and chromaticity diagram of 1960
From:www.adobe.com/. . . /cieluv.html



Equally perceived color distances and chromaticity diagram of 1976
From:www.adobe.com/. . . /cieluv.html

1.3 Color Temperature

radiation.html

In physics PLANCK'S LAW describes the electromagnetic radiation emitted from a so called BLACK BODY RADIATOR at a given temperature. A blackbody radiator is assumed to have very weak interaction with the surrounding environment and to be in a state of equilibrium. Stars are sufficiently good approximations of blackbody

radiators.

$$E(\lambda, T) = \frac{2hc^2}{\lambda^5} \frac{1}{e^{\frac{hc}{\lambda kT}} - 1}$$

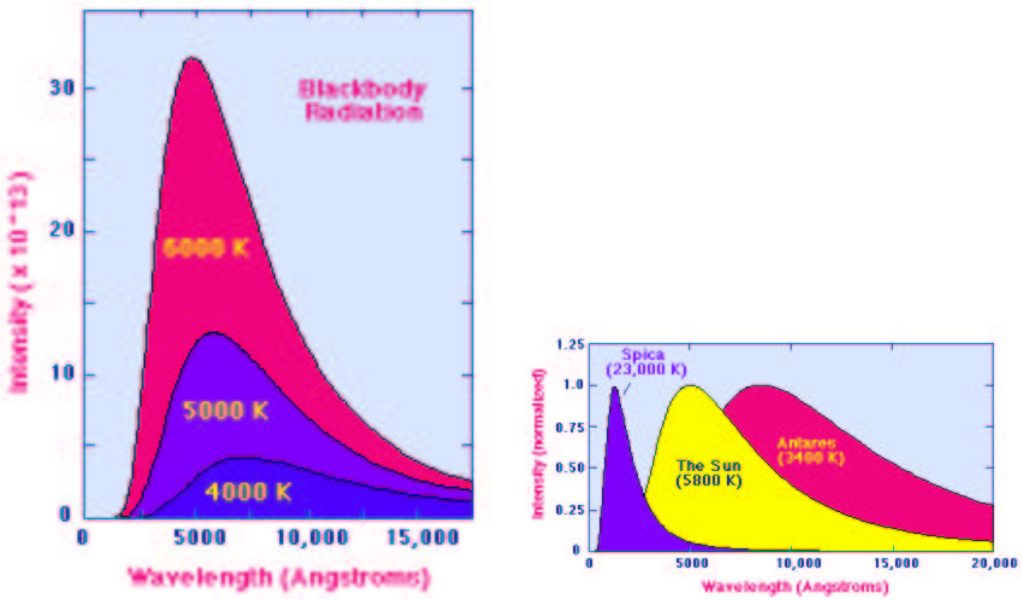
$h := 6.625 \times 10^{-27}$ erg-sec ... Planck constant

$k := 1.38 \times 10^{-16}$ erg/K ... Boltzmann constant

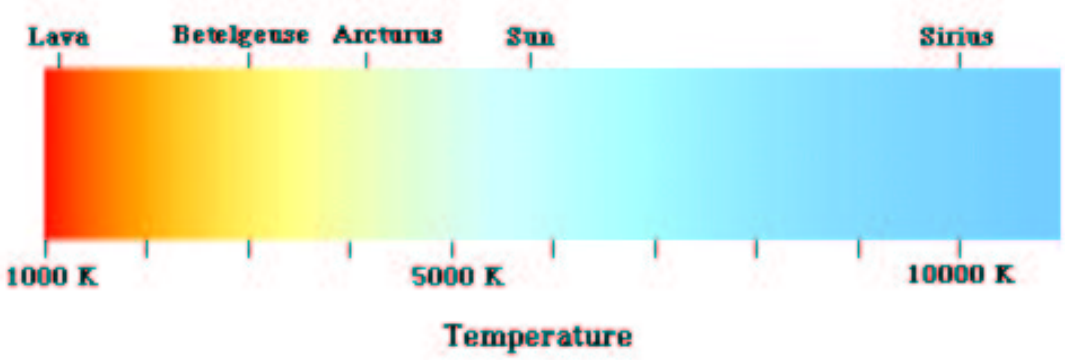
$C := 3 \times 10^{10}$cm/sec... speed of light

E ... emitted energy by unit surface area into a fixed direction

λ ... wavelength, T ... temperature in Kelvin.

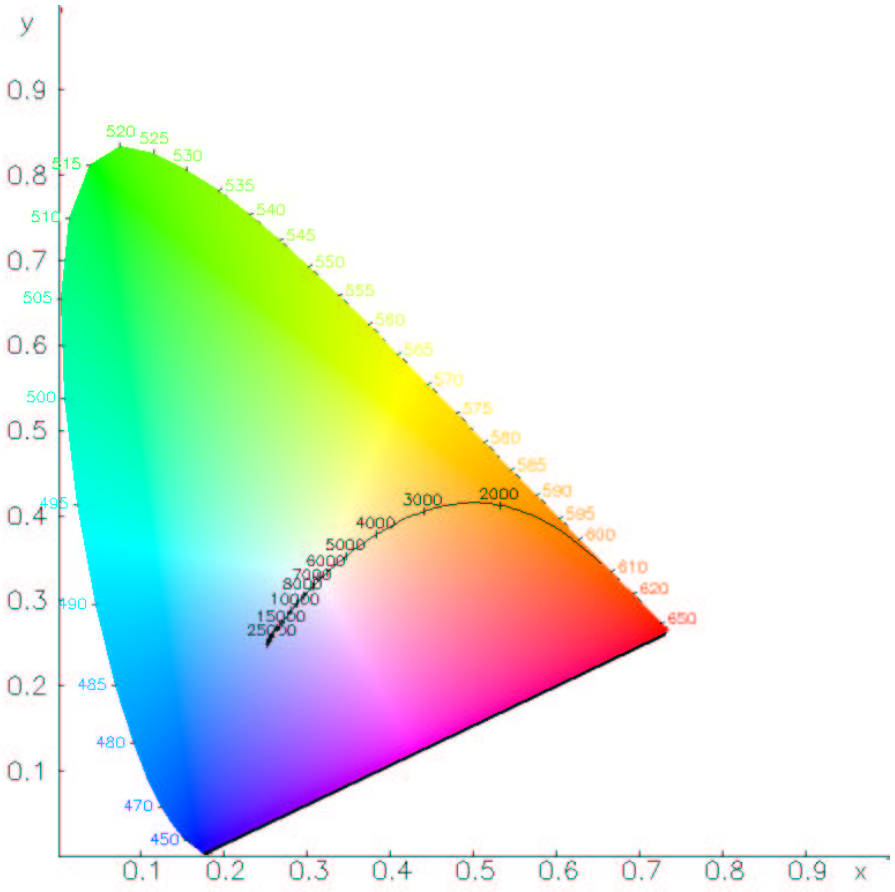


Energy distributions of black body and 3 stars



Surface temperatures of some stars, see plabpc.csustan.edu.html

Classification	Surface Temperature	Color	Familiar Examples
O	$30{,}000^oK$	electric blue	
B	$20{,}000^oK$	blue	Rigel
A	$10{,}000^oK$	white	Vega, Sirius
F	$7{,}000^oK$	yellow-white	Canopus
G	$6{,}000^oK$	yellow	Sun, Alpha Centauri
K	$4{,}000^oK$	orange	Arcturus, Aldebaran
M	$3{,}000^oK$	red	Betelgeuse, Barnard's Star



Positions of colors from black body radiator at different temperature

Note that many colors (like green and violet) are not lying on this curve, thus have no associated color temperature. Color temperature plays also an important role in photography. Lower temperature means redder; higher temperature means bluer. Household incandescent lighting has a relatively low color temperature-about 3,000° K. Open flames-candles, campfires-are still lower. By contrast, direct sunlight has a color temperature of about 5,400° K. Photographic film can not compensate for the color of ambient light like our visual system does. Usual

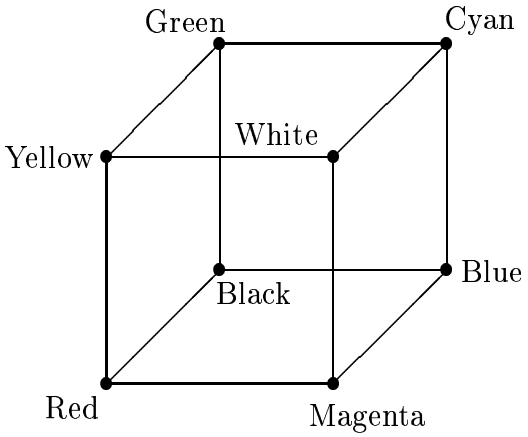
daylight film reproduces color most accurately at 5,400° K. If it is used inside (without flashlight) the taken pictures will look orange.

1.4 Color models for computer graphics

A COLOR MODEL is a specification of a color coordinate system and the subset of visible colors in this coordinate system. Conversion formulas between the color models have to be given.

1.4.1 The RGB Color Model

The red-green-blue model is formed by a color cube $\{(R, G, B) : 0 \leq R, G, B \leq 1\}$.



The RGB-cube

Conversion from (R, G, B) to (X, Y, Z) is given via the chromaticities (X_r, Y_r, Z_r), (X_g, Y_g, Z_g) and (X_b, Y_b, Z_b) of the CRTs phosphors by matrix multiplication via:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} Xr & X_g & X_b \\ Yr & Y_g & Y_b \\ Zr & Z_g & Z_b \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Let $C_r := X_r + Y_r + Z_r$. Then $X_r = x_r \cdot C_r$, $Y_r = y_r \cdot C_r$ and $Z_r = z_r \cdot C_r = (1 - x_r - y_r) \cdot C_r$.

This can be calculated from $X = \frac{x}{y}Y$, $Y = Y$, $Z = \frac{1-x-y}{y}Y$.

1.4.2 The CMY Color Model

This stands for cyan–magenta–yellow and is used for hardcopy devices. In contrast to color on the monitor, the color in printing acts subtractive and not additive. A printed color that looks red absorbs the other two components G and B and reflects R. Thus its (internal) color is G+B=CYAN. Similarly R+B=MAGENTA and R+G=YELLOW. Thus the C-M-Y coordinates are just the complements of the R-G-B coordinates:

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

If we want to print a red looking color (i.e. with R-G-B coordinates (1,0,0)) we have to use C-M-Y values of (0,1,1). Note that M absorbs G, similarly Y absorbs B and hence $M + Y$ absorbs all but R.

Black ($(R, G, B) = (0, 0, 0)$) corresponds to $(C, M, Y) = (1, 1, 1)$ which should in principle absorb R, G and B. But in practice this will appear as some dark gray. So in order to be able to produce better contrast printers often use black as 4^{th} color. This is the CMYK-model. Its coordinates are obtained from that of the CMY-model by $K := \max(C, M, Y)$, $C := C - K$, $M := M - K$ and $Y := Y - K$.

1.4.3 The YIQ Color Model

This is used for color TV. Here Y is the luminance (the only component necessary for B&W-TV). The conversion from RGB to YIQ is given by

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

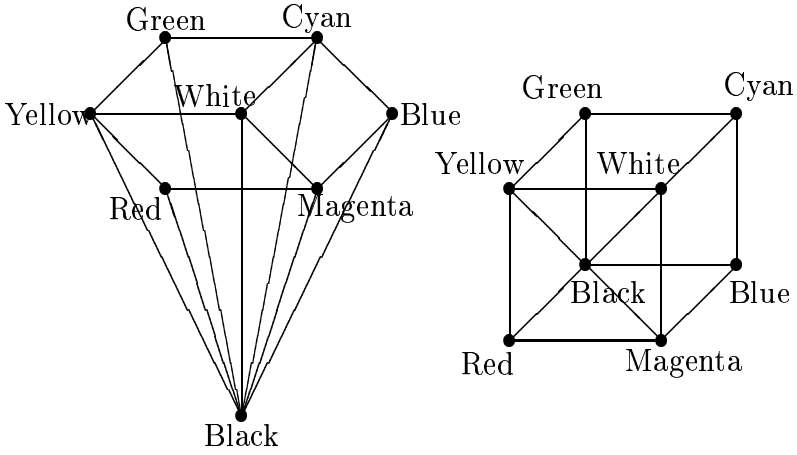
for standard NTSC RGB phosphor with chromaticity values

	R	G	B
x	0.67	0.21	0.14
y	0.33	0.71	0.08

The advantage of this model is that more bandwidth can be assigned to the Y-component (luminance) to which the human eye is more sensible than to color information. So for NTSC TV there are 4 MHz assigned to Y, 1.5 MHz to I and 0.6 MHz to Q.

1.4.4 The HSV color model

All color models treated so far are hardware oriented. The Hue-Saturation-Value model is oriented towards the user/artist. The allowed coordinates fill a six sided pyramid the 3 top faces of the color cube as base. Note that at the same height colors of different perceived brightness are positioned. Value is given by the height, saturation is coded in the distance from the axes and hue by the position on the boundary.

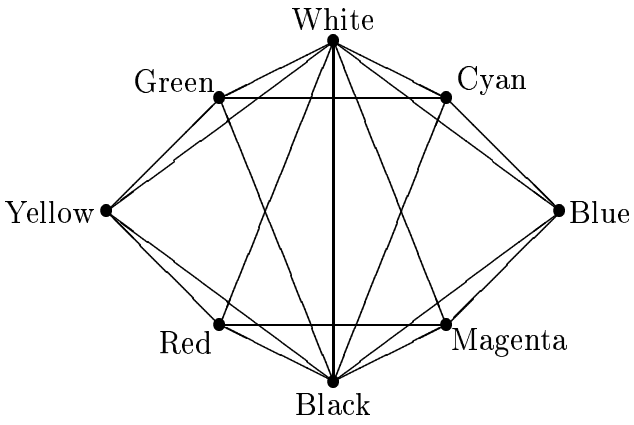


The HSV-model versus the RGB-model

Note that conversion from RGB to HSV is given by affine coordinate changes on each of the 3 four-sided sub-pyramids corresponding each to 1/3 of the color cube.

1.4.5 The HLS Color Model

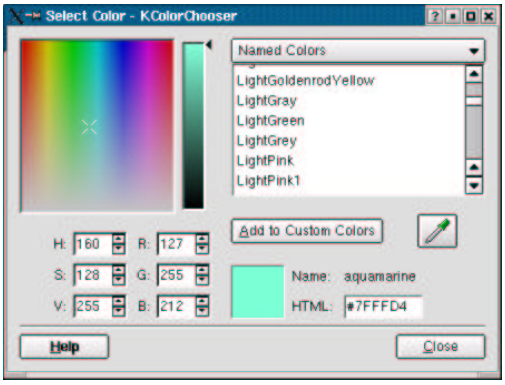
Here the RGB-cube is deformed in such a way that a six sided double pyramid results with the same base as in the HSV-model, but with two tips at black and at white.



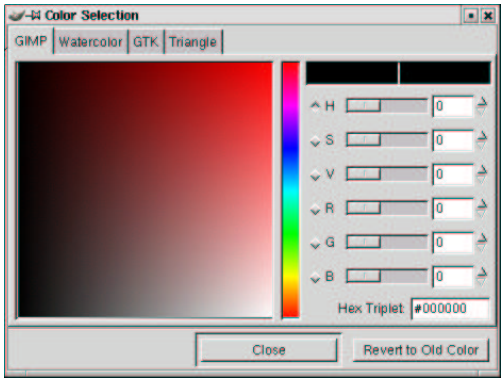
The HLS-model

1.4.6 Interactive Specification of Color

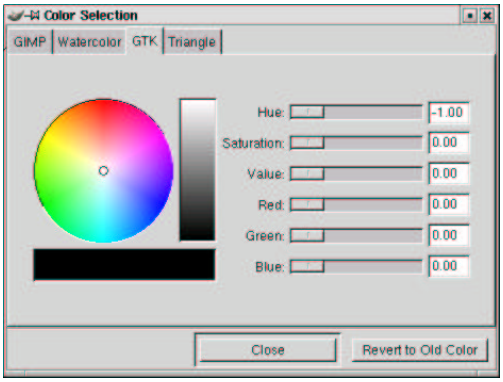
Color selectors for GUIs can be represented in various ways, e.g.:



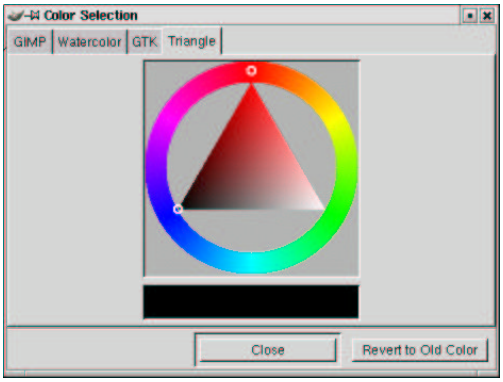
kcolor



gimp



GTK



gimp-triangle

1.4.7 Interpolating in Color Space

If we interpolate between two colors C_1 and C_2 then the result depends on the color model. Only where the conversion formulas are linear it does not matter which of the two models we use.

1.4.8 Using Color in Computer Graphics

There could be given many advices concerning the use of color in graphics design. E.g. using color for text web-pages should not be done to extensively. Since here we are mainly concerned with photo realistic images, this is not so much a topic here.

Chapter 2

Rastering

Literature

- Aliasing, Drawing Lines and Circles:
  [FvDFH90, 3.2], [PK87, 3], hugo elias: line drawing, hugo elias: scan converting
- Anti-aliasing:
  [FvDFH90, 3.17]
- Clipping:
  [FvDFH90, 3.11], [PK87, 5], ucdavis: clipping

2.1 Scan Converting Points

Cf. [PK87, 3.2].

On the screen we only have pixels with integer coordinates (in some range) only. So in order to display points with real coordinates, we have to round or truncate them. This is called rastering (dt: Rasterkonvertierung).

2.2 Scan Converting Lines

Cf. [FvDFH90, 3.2], [PK87, 3.3], and almond.srv.cs.cmu.edu/. . . /scanconv_2.pdf.

A line from point (x_0, y_0) to point (x_1, y_1) is mathematically given in explicit form by

$$y = k\,x + d, \text{ where the slope is } k := \frac{y_1 - y_0}{x_1 - x_0} \text{ and } d := y_0 - k\,x_0.$$

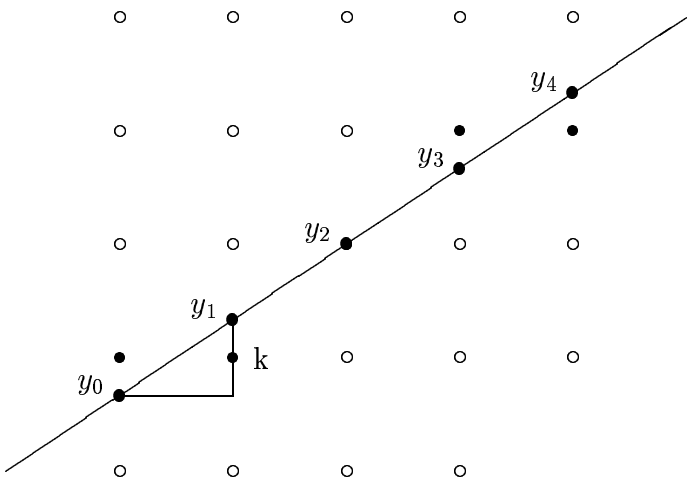
When we draw the line on the screen, we have to pick points with integer coordinates as close as possible to the line. If the slope satisfies $-1 \leq k \leq 1$, then we should pick in any column with given integer value x one point as close as possible to the corresponding point on the line. Its y-coordinate is given by $\bar{y} = \text{ROUND}(k\,x + d)$. If the slope satisfies $|k| > 1$ we may exchange x and y, in order to reduce the problem to the particular case. This method by brute force is inefficient because of the multiplication and the function ROUND.

2.2.1 Basic Incremental Algorithm

Cf. [FvDFH90, 3.2.1].

A better way to do this is recursively: The points on the line $y = k\,x + d$ for $x_i := x_0 + i$ are given inductively by

$$y_{i+1} := y_i + k.$$



Basic incremental algorithm for scan converting lines

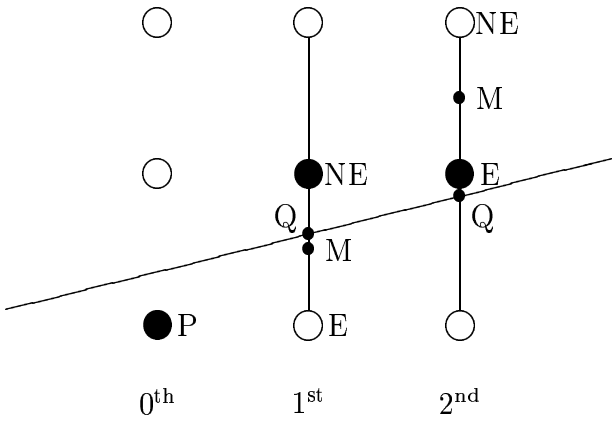
This way we get rid of the multiplication, but we still have to apply ROUND. Note however, that successive addition of a real number can lead to a cumulative error buildup. Here, this can usually be ignored.

2.2.2 Midpoint Line Algorithm

Cf. [FvDFH90, 3.2.2]

The midpoint line algorithm is due to Bresenham [Bre65] and was modified by Pitteway [Pit67] and Van Aken [VA84]. It works as follows: Let the slope of the line be $0 \leq k \leq 1$. Suppose one approximate point $P = (x, \bar{y})$ is already determined. We have only two choices for the next point, namely $E := (x+1, \bar{y})$ and $NE := (x+1, \bar{y}+1)$ and we should choose the one which is closer to $k(x+1)+d$. To determine the appropriate choice we proceed as follows:

- calculate the middle point $M := (x+1, \bar{y}+\frac{1}{2})$.
- If the intersection point Q of the line with the vertical line connecting E and NE is below M, take E as next pixel.
- Otherwise take NE as next pixel.



Midpoint line algorithm

In oder to check this condition we consider the implicit equation

$$f(x,y) := a\,x + b\,y + c := (y_1 - y_0)\,x - (x_1 - x_0)\,y + (x_1\,y_0 - y_1\,x_0)$$

where we may assume that $a > 0$. Note that $f(x,y) = 0$ if and only if (x,y) lies on the line. And $f(x,y) > 0$ if and only if (x,y) lies below the line. So in the first step we have to test

$$\begin{aligned} f\left(x+1, \bar{y}+\frac{1}{2}\right) &= a\,(x+1) + b\left(\bar{y}+\frac{1}{2}\right) + c \\ &= (a\,x + b\,\bar{y} + c) + \left(a + b\,\frac{1}{2}\right) = f(x,\bar{y}) + a + \frac{b}{2}. \end{aligned}$$

In case we choose E we have to test for the next (2^{nd}) column

$$f\left(x+2, \bar{y}+\frac{1}{2}\right) = f\left(x+1, \bar{y}+\frac{1}{2}\right) + a.$$

In case we choose NE we have to test

$$f\left(x+2, \bar{y}+1+\frac{1}{2}\right) = f\left(x+1, \bar{y}+\frac{1}{2}\right) + a + b.$$

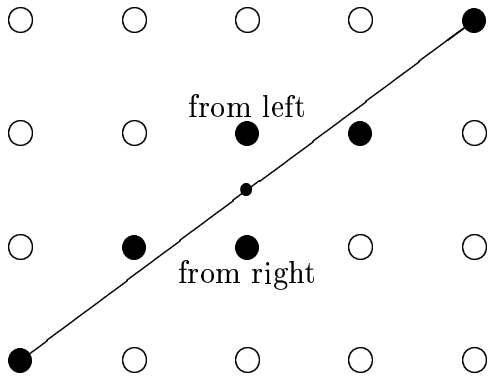
Thus the test for the next column can be easily obtained from that for the previous one by adding a or $a+b$ accordingly.

Lines with other slopes can be obtained by mirroring.

Some issues concerning this algorithm have to be taken into account, cf. [FvDFH90, 3.2.3]:

- Endpoint order: This procedure is not independent from which side of the line we are starting from (in case the line hits some middle point). It is not a good idea to sort the endpoints first in case of line-styles like dashing, since a polygon should be drawn in succession.

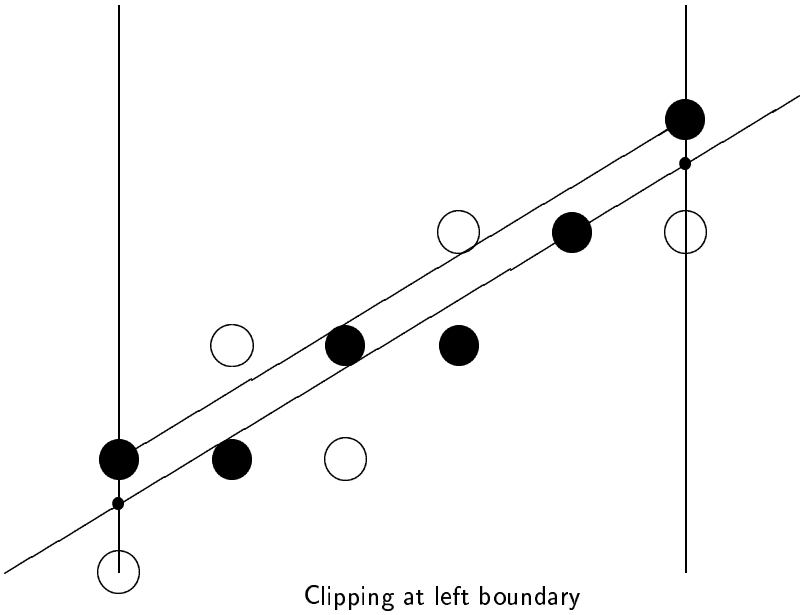
  E.g., the following line from $(0,0)$ to $(4,3)$:



Dependency on order of endpoints

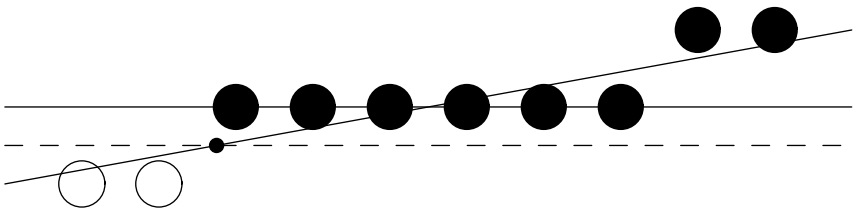
- Starting at the clipping edge:

  Do not clip (see (2.5) below) and then scan-convert if line hits the left boundary of the clipping rectangle.



Clipping at left boundary

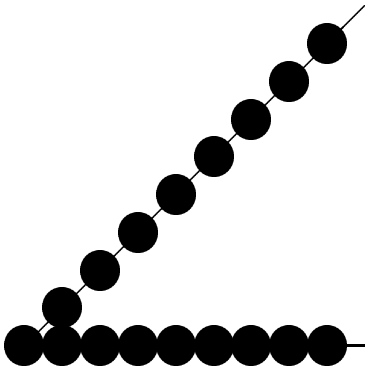
In case of a horizontal boundary we may miss some points at the beginning. In oder to avoid this we should start at the intersection point with the line $y = y_{\min} - \frac{1}{2}$ below the horizontal lower boundary $y = y_{\min}$



Clipping at the lower boundary

- Varying Intensity:

  The number of pixel per length depends on the slope. It will be 1 for horizontal lines and $1/\sqrt{2}$ for slope 1.



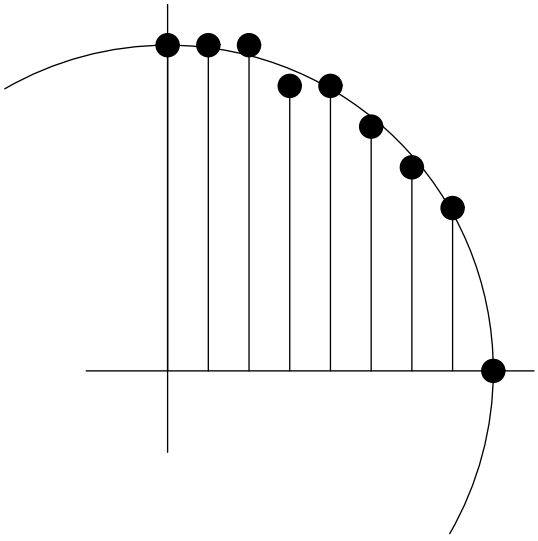
Dependency of intensity on slope of line

So one could modify the intensity of pixels drawn. Another (and even better method) is anti-aliasing as discussed below.

2.3 Scan Converting Circles

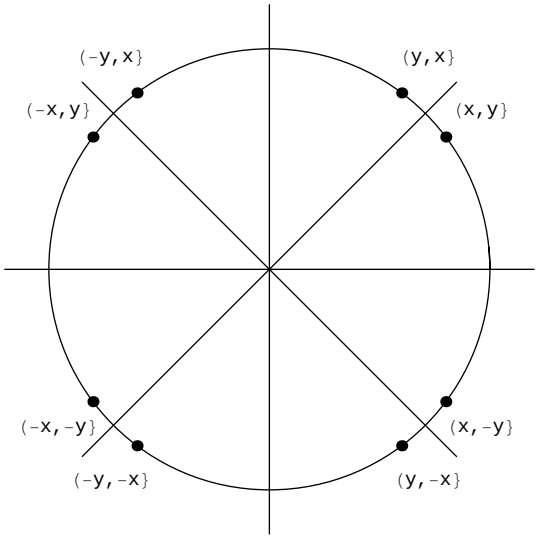
Cf. [FvDFH90, 3.3] and [PK87, 3.4]

A circle (through 0) with radius R is given by the explicit equation $y = \pm\sqrt{R^2 - x^2}$ or implicitly by $0 = F(x, y) := x^2 + y^2 - R^2$. The straight forward method of drawing a circle by approximating the values $\pm\sqrt{R^2 - i^2}$ is ineffective (since it involves: squaring, taking roots and ROUND) and it gives an asymmetric distribution.



Straight forward scan converting a circle

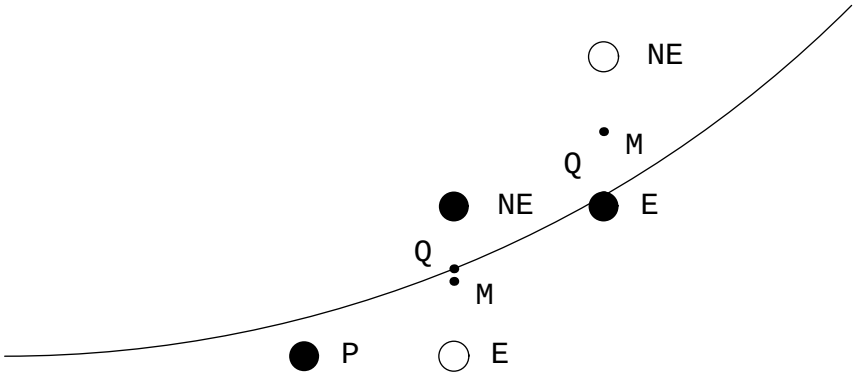
We can make use of the 8-fold symmetry, so we only have to draw 1/8 of the circle say from S to SE.



8-fold symmetry of the circle

2.3.1 Midpoint Circle Algorithm

Cf. [FvDFH90, 3.3.2].



Midpoint circle algorithm

The midpoint circle algorithm of Bresenham [Bre77] and [BGP83] is analogously to that of straight lines and goes as follows:

$$F\left(x+1, \bar{y}+\frac{1}{2}\right) = (x+1)^2 + \left(\bar{y}+\frac{1}{2}\right)^2 - R^2 = F(x,\bar{y}) + 2x + \bar{y} + \frac{5}{4}$$

and in the next step:

$$E: \quad F\left(x+2, \bar{y}+\frac{1}{2}\right) = F\left(x+1, \bar{y}+\frac{1}{2}\right) + 2x + 3$$
$$NE: \quad F\left(x+2, \bar{y}+\frac{3}{2}\right) = F\left(x+1, \bar{y}+\frac{1}{2}\right) + 2x + 2\bar{y} + 5$$

One can speed up things be calculating the linear increments recursively.

2.4 Scan Converting Ellipses

Cf. [FvDFH90, 3.4] [PK87, 3.5]

Ellipses no longer have an 8-fold symmetry. So we have to determine the point where the slope is 1. Differentiating the implicit equation $0 = F(x,y) = b^2x^2 + a^2y^2 - a^2b^2$ gives

$$0 = F_x(x,y) + F_y(x,y)\, y' = 2b^2\, x + 2a^2\, y\, y',$$

so we have $y' = 1$ when $a^2\, y + b^2\, x = 0$ and inserting this into the implicit equation gives $a^2b^2x^2 + b^4\, x^2 - a^4b^2 = 0$, i.e. $x = \pm\frac{a^2}{a^2+b^2}$.

2.5 Clipping Lines

Cf. [FvDFH90, 3.12] and [PK87, 5.3].

Often clipping beforehand is not advisable, in particular, if the primitive extends beyond the clipping region only a little. In this case one should determine all the approximation points, but draw only those inside the clipping region.

2.5.1 Brute Force

cf. [FvDFH90, 3.12.2]

To clip a point (x,y) against a rectangle $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ means to draw the point exactly if the following condition is satisfied:

$$(x_{\min} \le x \le x_{\max}) \text{ and } (y_{\min} \le y \le y_{\max}).$$

Clipping a line against a rectangle can lead to one of the following cases:

- Both endpoints lie inside. Hence the whole segment is inside and is to be drawn completely.
- Exactly one endpoint lies inside. Hence the line hits boundary exactly once and we have to determine the corresponding intersection point and clip the line there.
- Both endpoints lie outside. Then it is not clear whether the line hits the rectangle or not. So we intersect this line with each of the 4 sides (i.e. the infinite lines) and check whether some intersection points lies on the boundary of the rectangle. This is best done in parametric form

$$x = x_0 + t(x_1 - x_0) \quad y = y_0 + t(y_1 - y_0).$$

But this is still an inefficient method.

2.5.2 Cohen-Sutherland Line-Clipping Algorithm

Cf. [FvDFH90, 3.12.3].

First we test whether both endpoints are inside (and hence draw the line segment) or whether both are left of $x = x_{\min}$, right of $x = x_{\max}$, below $y = y_{\min}$, or above $y = y_{\max}$ (then we ignore line segment). Otherwise we split the line segment into two pieces at a clipping edge (and thus reject one part). Now we proceed iteratively.

A rather simple accept–reject test is the following:

Divide the plane into 9 regions and assign a 4 bit code to each:

1000 ... above top edge $y > y_{\max}$

0100 ... below bottom edge $y < y_{\min}$

0010 ... right of right edge $x > x_{\max}$

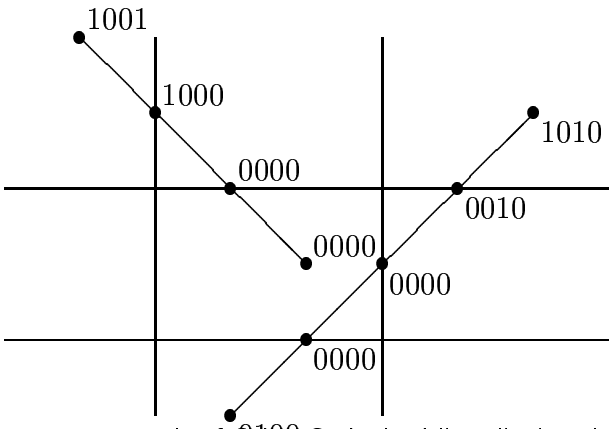
0001 ... left of left edge $x < x_{\min}$

mathcalculate the corresponding bit-codes for both endpoints.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Codes for the 9 regions associated to clipping rectangle

If both codes are zero then the line segment is completely inside the rectangle. If the bitwise-and of these codes is not zero then the line does not hit since both endpoints lie on the wrong side of at least one boundary line (corresponding to a bit equal to 1). Otherwise take a line which is met by the segment (for this find one non-zero bit), divide the given line at the intersection point in two parts and reject the one lying in the outside halfplane.

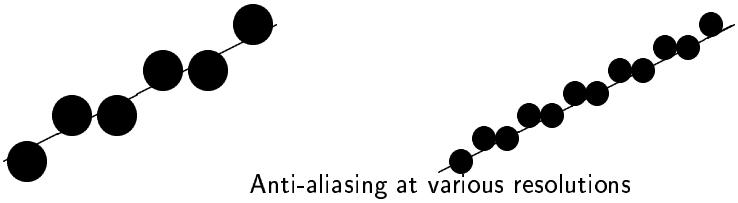


Example of Cohen-Sutherland line-clipping algorithm

2.6 Anti-aliasing

Cf. [FvDFH90, 3.17] and [PK87, 3.10].

Aliasing lines with slope $k \neq 0$ leads to ugly looking jagged edges also called staircasing. We could improve the appearance by increasing the resolution, but this is rather memory- and time-consuming.

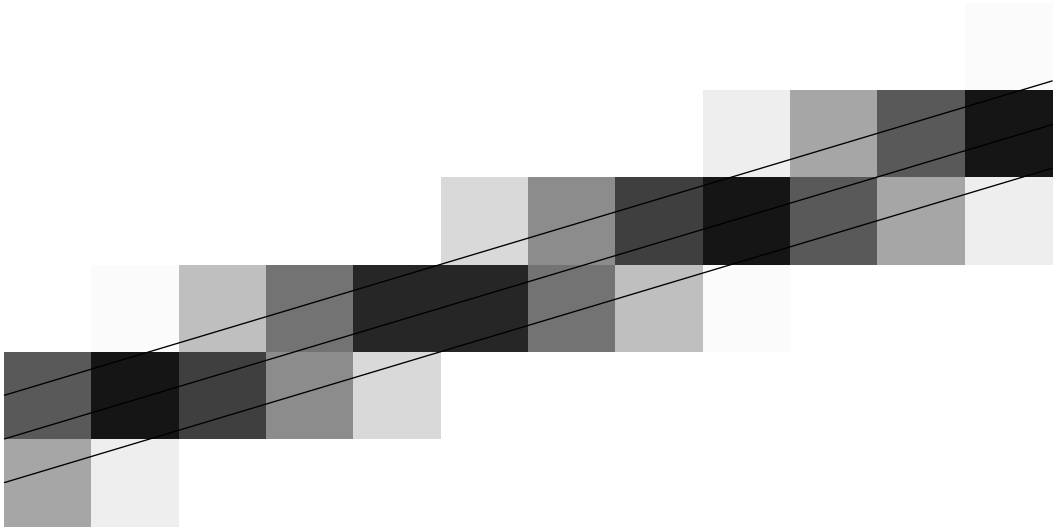


Anti-aliasing at various resolutions

2.6.1 Unweighted Area Sampling

cf. [FvDFH90, 3.17.2]

Here we consider instead of the mathematically infinitely thin line a line with some thickness, i.e. a small rectangle. We color each pixel according to the area of the corresponding unit square which is covered by this rectangle.



Antialiasing by unweighted area sampling

2.6.2 Weighted Area Sampling

Cf. [FvDFH90, 3.17.3].

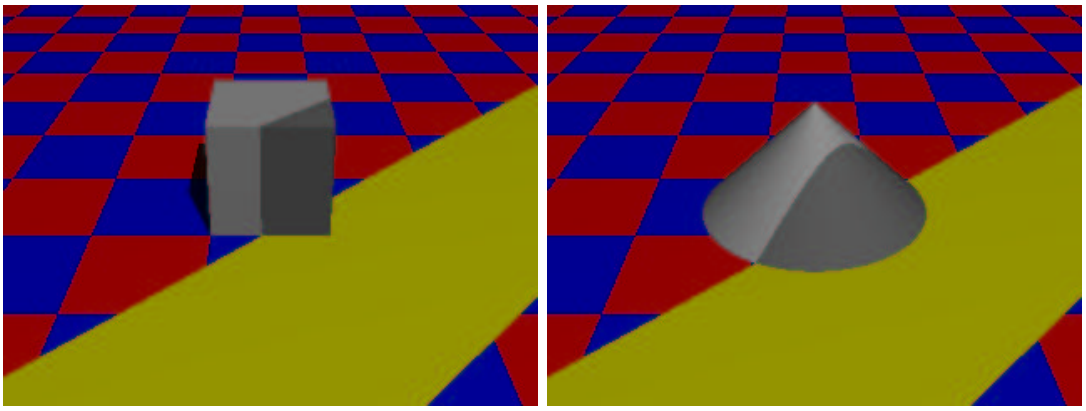
An even better method is to color each pixel according to area of the corresponding square which is covered by the rectangle but weighted according to distance from the center of the pixel.

A particular wighting function is used by the

2.6.3 Gupta-Sproull Anti-aliasing Algorithm

cf. [FvDFH90, 3.17.4]

This algorithm weights the intersecting area, by considering a cone with radius 1 (hence a line with slope 1 intersects 3 vertical pixels) and take the volume of the part of this cone lying above the rectangle representing the line. To speed things up, we need not calculate these volumina during runtime but can use a fixed pre-calculated table of intensities depending on the various distances to the center. We can use the scan-conversion algorithm to determine the approximation points, but set the intensity levels of then and the two vertically neighboring pixels according to their distance.



Weighting function of plain- and of Gupta-Sproull antialiasing

Chapter 3

File-formats

Literature

- tweedy & reddy: graphics file format page
  www.dcs.ed.ac.uk/. . . /index-hi.html

Some graphic formats (alphabetically sorted):

BMP . . . MS-Windows Bitmap

EPSF . . . Encapsulated Postscript

GIF . . . Graphics Interchange Format

IFF . . . Interchange File Format

JP(E)G . . . Joint Photographic Experts Group

PNG . . . Portable Network Graphics

PNM=PBM+ . . . enhanced portable bitmap

- **PBM** . . . Monochrome bitmap
- **PGM** . . . Grayscale bitmap
- **PPM** . . . Full color images
- **PNM** . . . Anymap

TGA . . . Targa File Format

TIFF ... Tag Image File Format

XPM ... X Pixmap Format

Now in more detail some selected formats:

3.1 RAW-data

For black&white pictures we need the following Information: width, height, and one bit per pixel: So a 800×600 picture is $800 \times 600/8 = 60000B = 60KB$, A 1024×768 picture is $1024 \times 768/8 = 98304 \approx 100KB$. A 2272×1704 picture (3.8 Mio Pixel) is $483936 \approx 500KB$.

If we use a color palette of fixed colors (e.g. gray levels) out of 256 (1Byte), then we need one byte per pixel.

If we code the colors as RGB-values with 5bits (32 levels) per color, then we need 2 bytes (minus 1bit) per pixel.

If we code the colors as RGB-values with 8bits per color, then we need 3 bytes per pixel.

If we code the colors as RGB-values with 12bits per color (4096 levels), then we need 36bit=4.5bytes per pixel.

If we code the colors as RGB-values with 16bits per color (65536 levels), then we need 48bit=6bytes per pixel.

width	height	pixel	bits/pixel	colors	size in bytes
w	h	w*h	d	2^d	w*h*d/8
640	480	307200	1	2	38400
640	480	307200	8	256	307200
640	480	307200	16	65536	614400
640	480	307200	24	16777216	921600
640	480	307200	36	6.8719e+10	1382400
640	480	307200	48	2.8147e+14	1843200
800	600	480000	1	2	60000
800	600	480000	8	256	480000
800	600	480000	16	65536	960000
800	600	480000	24	16777216	1440000
800	600	480000	36	6.8719e+10	2160000

800	600	480000	48	2.8147e+14	2880000
1024	768	786432	1	2	98304
1024	768	786432	8	256	786432
1024	768	786432	16	65536	1572864
1024	768	786432	24	16777216	2359296
1024	768	786432	36	6.8719e+10	3538944
1024	768	786432	48	2.8147e+14	4718592
2272	1704	3871488	1	2	483936
2272	1704	3871488	8	256	3871488
2272	1704	3871488	16	65536	7742976
2272	1704	3871488	24	16777216	11614464
2272	1704	3871488	36	6.8719e+10	17421696
2272	1704	3871488	48	2.8147e+14	23228928

3.2 PNM. Portable Anymap File Format

This is a very simple format, which is easily and quickly read and written. It comes in three flavors PBM, PGM and PPM and each of this has a plain and a raw version.

3.2.1 PBM. Portable Bit-Map Format

See the man-page.

Plain version:

- Identifier "P1"
- ASCII representation of Width
- ASCII representation of Height
- Height many lines of Width many pixels as ASCII characters separated by space.

Raw version:

- Identifier "P4"

- ASCII representation of Width
- ASCII representation of Height
- Height many lines of Width many pixels (8 pixel packed in one byte, padded at end of line) from left to right and top to bottom (high bit first)
- This uses 2byte=16bit/pixel.

3.2.2 PGM. Portable Grey-Map Format

See the man-page.

Plain version:

- Identifier "P2"
- ASCII representation of Width
- ASCII representation of Height
- ASCII representation of Maxvalue ($<65536 = 2^{16}$)
- Height many lines of Width many pixels as ASCII words 0..Maxvalue separated by space

Raw version:

- Identifier "P5"
- ASCII representation of Width
- ASCII representation of Height
- ASCII representation of Maxvalue ($<65536 = 2^{16}$)
- Height many lines of Width many pixels as one or two bytes from left to right and top to bottom

3.2.3 PPM. Portable Pix-Map Format

See the man-page.

Plain version:

- Identifier "P3"
- ASCII representation of Width
- ASCII representation of Height
- ASCII representation of Maxvalue ($<65536 = 2^{16}$) per color component
- Height many lines of Width many pixels as 3 ASCII words 0..Maxvalue (R/G/B) separated by space

Raw version:

- Identifier "P6"
- ASCII representation of Width
- ASCII representation of Height
- ASCII representation of Maxvalue ($< 5536 = 2^{16}$)
- Height many lines of Width many pixels as 3 bytes or double-bytes (R/G/B) from left to right and top to bottom. The colors are coded non-linear with a gamma of 2.2.

3.3 Compression

Cf.

- www.ztt.fh-worms.de/.../index.html Lecture Notes (in German).
- www.cs.sfu.ca/.../index.html Nice short description and Java applets.
- www.faqs.org/.../index.html Compression FAQ.
- almond.srv.cs.cmu.edu/.../compression.pdf Paul Heckbert's Computer Graphic Lecture Notes.

In order to use less disk space and bandwidth during transmission we have to compress the image using some packing algorithm.

3.3.1 Run-length encoding

Run-Length-Encoding is a simple and fast procedure for coding symbols which appear often in succession. For example a line/vector graphic (with few lines) will contain mainly pixel in the background color. Each repeating symbol is coded once together with the repetition count.

We consider two flavors of this coding:

7-Bit-Run-Length-Encoding:

If the symbols are (mainly) in the range 00h–7Fh (like it is the case with ASCII-text) we code repeating symbols by first writing the count ($< 80h$) with highest bit set to 1 followed by the symbol. For symbols with highest bit 1 (like umlauts or other accented symbols) we have to precede them by $81h$, since this is not a useful count indicator:

[0|symbol] or [1|count][0|symbol] or [1|0000001][symbol]

Standard-Run-Length-Encoding:

In the standard run-length-encoding one codes multiple appearing bytes by three bytes: the original byte, 90h, and the count. Since the count has to be $< 80h$ longer chains of the same symbol have to be coded by multiple of such triples. For the symbol 90h another coding has to be used, since 90h in the output means 'count follows'. So 90h will be coded as 90 00h, since this is does not represent a reasonable count.

Some coding procedures apply first run-length-encoding before the proper compression algorithm is applied.

Pseudo-code for the general run-length-algorithm looks as follows:

```
  Loop: count = 0
        REPEAT
          get next symbol
          count = count + 1
        UNTIL (symbol unequal to next one)
output symbol
        IF count > 1
          output count
        GOTO Loop
```

3.3.2 Huffman

Cf.

- www.ztt.fh-worms.de/.../node9.html
- DC-Sec3.html
- www.cs.sfu.ca/.../Huffman.html
- www.cs.sfu.ca/.../AdaptiveHuff.html

Basic Huffman encoding goes as follows:

- Count the number of times each symbol appears.
- Now built a binary tree by assigning each symbol to a node and repeating connecting nodes in the following way until all nodes are connected:
  - Find two nodes with lowest counts.
  - Create a parent node for them with the sum of their counts as count and connect the two children to the parent node.
- Now every symbol is encoded by starting at the root of the tree and appending 0 or 1 depending whether you have to go right or left to reach the node of the symbol.
- Note that since all the symbols are at the leafs (the ends of the branches) of the tree, there is never a chance that one code will be the prefix of another one. This property ensures that decoding can be done by collecting bits from the coded string until the corresponding sequence reaches a leaf and output the associated symbol.

There are a few shortcomings to the basic Huffman compression. First of all, one has to send the Huffman tree at the beginning of the compressed file, or the decompressor will not be able to decode it.

Furthermore, Huffman compression looks at the statistics of the whole file, so that if a part of the code uses some character more frequently, no adjustment is made. Even worse, sometimes (like for live information) the whole input stream is not available when compression should start.

Because of these reasons adaptive Huffman encoding has been invented, but we will not treat this topic.

3.3.3 LZW. Lempel Ziv Welch

Cf.

- www.ztt.fh-worms.de/. . . /node17.html
- www.cs.sfu.ca/. . . /LZW.html
- www.dogma.net/. . . /lzw.htm

From www.cs.sfu.ca/. . . /LZW.html:
LZW compression has its roots in the work of Jacob Ziv and Abraham Lempel. In 1977, they published a paper on "sliding-window" compression, and followed it with another paper in 1978 on "dictionary" based compression. These algorithms were named LZ77 and LZ78, respectively. Then in 1984, Terry Welch made a modification to LZ78 which became very popular and was dubbed LZW (guess why). The LZW algorithm is what we are going to talk about here.

The Concept:

Many files, especially text files, have certain strings that repeat very often, for example "the" in english text. With the spaces, the string takes 5 bytes, or 40 bits to encode. But what if we were to add the whole string to the list of characters after the last one, at 256. Then every time we came across "the", we could send the code 256 instead of 32,116,104,101,32. This would take 9 bits instead of 40 (since 256 does not fit into 8 bits).

This is exactly the approach that LZW compression takes. It starts with a "**dictionary**" of all the single character with indexes 0..255. It then starts to expand the dictionary as information gets sent through. Pretty soon, redundant strings will be coded as a single bit, and compression has occured.

The Algorithm:

Ok, so how is this done? Here is the basic algorithm:

```
STRING = get input character
WHILE there are still input characters DO
    CHARACTER = get input character
    IF STRING+CHARACTER is in the string table then
        STRING = STRING+character
    ELSE
        output the code for STRING
        add STRING+CHARACTER to the string table
```

```
        STRING = CHARACTER
    END of IF
END of WHILE
output the code for STRING
```

So what happens here? The program reads one character at a time. If the code is in the dictionary, then it adds the character to the current work string, and waits for the next one. This occurs on the first character as well. If the work string is not in the dictionary, (such as when the second character comes across), it adds the work string to the dictionary and outputs the work string without the new character. It then sets the work string to the new character.

How about decompression?

Note that when a new entry is added to the dictionary, then its root is given by the code that is being sent and the suffix character is the first chacter for the code which is send next.

So in order to decompress we have to build up the dictionary by adding a new entry every time a code is received. The entries root is just the translation of the code being received and its suffix character is the first character of the translation of the next code which gets received.

```
Read OLD_CODE
output OLD_CODE
WHILE there are still input characters DO
    Read NEW_CODE
    STRING = get translation of NEW_CODE
    output STRING
    CHARACTER = first character in STRING
    add OLD_CODE + CHARACTER to the translation table
    OLD_CODE = NEW_CODE
END of WHILE
```

There is one problem, namely if the next code being sent is identically to the one just being added to the dictionary. So this happends if "root" was already found, "root"+'suffix' appears first and 'suffix' + what follows equals "root"+'suffix'. So the input looks like:

$$\underbrace{\text{suffix} + \text{string}}_{\text{root}} + \underbrace{\text{suffix} + \text{string}}_{\text{root}}$$

In this case the decoder doesn't know yet the translation of this code but since the its first character has to be equal to the first character of the current code, in can use this one instead.

```
CHARACTER = read first code
output CHARACTER
OLD_CODE = CHARACTER
WHILE there are still input characters DO
    NEW_CODE = read next code
    IF NEW_CODE is not in the translation table THEN
        STRING = get translation of OLD_CODE
        STRING = STRING + CHARACTER
    ELSE
        STRING = get translation of NEW_CODE
    END of IF
    output STRING
    CHARACTER = first character in STRING
    add OLD_CODE + CHARACTER to the translation table
    OLD_CODE = NEW_CODE
END of WHILE
```

The nice thing is that the decompressor builds its own dictionary on its side, that matches exactly the compressor's, so that only the codes need to be sent.

For an explanation concerning the gif variant see:
www.dcs.ed.ac.uk/. . . /GIF-comp.txt

3.4 GIF. Graphics Interchange Format

Cf.

- www.dcs.ed.ac.uk/. . . /GIF87a.txt
- www.dcs.ed.ac.uk/. . . /GIF89a.txt
- www.dcs.ed.ac.uk/. . . /GIF-comp.txt

This format is organized by a block structure.

- Header-block:
  - 'GIF' ... 3 byte identifier
  - version ... 3 bytes (e.g. '87a', '89a')

- Logical Screen Descriptor:
  - Logical Screen Width ... 2 bytes
  - Logical Screen Height ... 2 bytes
  - Flags ... 1 byte consisting of
    - Global Color Table Flag ... 1 bit
    - Color Resolution ... 3 bits
    - Color Table Sort Flag .. 1 bit
    - Size of Color Table ... 3 bits
  - Background Color Index ... 1 byte
  - Pixel Aspect Ratio ... 1 byte
- Global Color Table (if Global Color Table Flag is 1): Colors are coded by a selectable color-table of 256 entries. $3 \times 2^{(\text{Size of Color Table})}$ bytes of R/G/B information.
- Image Descriptor:
  - 0x2C ... 1 byte Image Separator
  - Image Left Pos ... 2 bytes
  - Image Top Pos .... 2 bytes
  - Image Width ..... 2 bytes
  - Image Height ..... 2 bytes
  - Flags ..... 1 byte consisting of Local Color Table Flag, Interlace Flag, Sort Flag, Reserved, Size Local Color Table
- Table Based Image Data (packed with LZW):
  - LZW minimum code ... 1 byte
  - Subblocks of at most 255 bytes each, these are LZW compressed with variable length output codes starting at codesize+1 (minimum 3 bits) and ending at 12 bits (Maximal code is thus 4095=0xFFF)

    Uses LZ78 (patented by Unisys), see below.

    Bits are packed from right to left. Then output is cut into subblocks of length ≤ 255.
  - Terminated by a zero length subblock
- Graphic Control Extension (Identifier: 0x21 0xF9)

- Comment Extension (Identifier: 0x21 0xFE)
- Plain Text Extension (Identifier: 0x21 0x01)
- Application Extension (Identifier: 0x21 0xFF)
- Trailer: 0x3B ... 1 byte

This format is best used for small images (e.g. icons), images with sharp edges, and images with few colors.

For details on LZ78 used by GIF see www.dcs.ed.ac.uk/.../GIF-comp.txt

3.5 PNG. Portable Network Graphics

Cf.

- www.libpng.org.html
- png-1.2-pdg.html

This is pronounced "ping". Colors can be coded up to 6*8=48 bits. It uses (non-patented) LZ77 compression.

> From png-1.2-pdg.html:
>
> - GIF features retained in PNG include:
>   - Indexed-color images of up to 256 colors.
>   - Streamability: files can be read and written serially, thus allowing the file format to be used as a communications protocol for on-the-fly generation and display of images.
>   - Progressive display: a suitably prepared image file can be displayed as it is received over a communications link, yielding a low-resolution image very quickly followed by gradual improvement of detail.
>   - Transparency: portions of the image can be marked as transparent, creating the effect of a non-rectangular image.
>   - Ancillary information: textual comments and other data can be stored within the image file.
>   - Complete hardware and platform independence.

  - Effective, 100% lossless compression.
- Important new features of PNG, not available in GIF, include:
  - Truecolor images of up to 48 bits per pixel.
  - Grayscale images of up to 16 bits per pixel.
  - Full alpha channel (general transparency masks).
  - Image gamma information, which supports automatic display of images with correct brightness/contrast regardless of the machines used to originate and display the image.
  - Reliable, straightforward detection of file corruption.
  - Faster initial presentation in progressive display mode.
- PNG is designed to be:
  - Simple and portable: developers should be able to implement PNG easily.
  - Legally unencumbered: to the best knowledge of the PNG authors, no algorithms under legal challenge are used. (Some considerable effort has been spent to verify this.)
  - Well compressed: both indexed-color and truecolor images are compressed as effectively as in any other widely used lossless format, and in most cases more effectively.
  - Interchangeable: any standard-conforming PNG decoder must read all conforming PNG files.
  - Flexible: the format allows for future extensions and private add-ons, without compromising interchangeability of basic PNG.
  - Robust: the design supports full file integrity checking as well as simple, quick detection of common transmission errors.

File Structure:

- Signature

(decimal)	137	80	78	71	13	10	26	10
(hexadecimal)	89	50	4e	47	0d	0a	1a	0a
(ASCII C notation)	\211	P	N	G	\r	\n	\032	\n

- Chunk layout
  - length . . . 4 bytes
  - chunk type . . . 4 bytes (chars)
  - data . . . length bytes
  - CRC (Cyclic Redundancy Check) . . . 4 bytes

- IHDR Image Header
  - Width: 4 bytes
  - Height: 4 bytes
  - Bit depth: 1 byte
  - Color type: 1 byte
  - Compression method: 1 byte
  - Filter method: 1 byte
  - Interlace method: 1 byte
- PLTE Palette
  - Red: 1 byte (0 = black, 255 = red)
  - Green: 1 byte (0 = black, 255 = green)
  - Blue: 1 byte (0 = black, 255 = blue)
- IDAT Image data
- IEND Image trailer

Optional Chunks:

- tRNS Transparency
- gAMA Image gamma
- cHRM Primary chromaticities
- sRGB Standard RGB color space
- iCCP Embedded ICC profile
- tEXt Textual data
- zTXt Compressed textual data
- iTXt International textual data
- bKGD Background color
- pHYs Physical pixel dimensions
- sBIT Significant bits
- sPLT Suggested palette
- hIST Palette histogram
- tIME Image last-modification time

About the motivation for introducing this new format:

From png-1.2-pdg.html:
We considered numerous existing formats before deciding to develop PNG. None could meet the requirements that we felt were important for PNG.

GIF is no longer suitable as a universal standard because of legal entanglements. Although just replacing GIF's compression method would avoid that problem, GIF does not support truecolor images, alpha channels, or gamma correction. The spec has more subtle problems too. Only a small subset of the GIF89 spec is actually portable across a variety of implementations, but there is no codification of the most portable part of the spec.

TIFF (the Tagged Image File Format) is far too complex to meet our goals of simplicity and interchangeability. Defining a TIFF subset would meet that objection, but would frustrate users making the reasonable assumption that a file saved as TIFF from their existing software would load into a program supporting our flavor of TIFF. Furthermore, TIFF is not designed for stream processing, has no provision for progressive display, and does not currently provide any good, legally unencumbered, lossless compression method.

IFF has also been suggested, but is not suitable in detail: available image representations are too machine-specific or not adequately compressed. The overall chunk structure of IFF is a useful concept that PNG has liberally borrowed from, but we did not attempt to be bit-for-bit compatible with IFF chunk structure. Again this is due to detailed issues, notably the fact that IFF FORMs are not designed to be serially writable.

Lossless JPEG is not suitable because it does not provide for the storage of indexed-color images. Furthermore, its lossless truecolor compression is often inferior to that of PNG.

From png-1.2-pdg.html:
Deflate/Inflate Compression

PNG compression method 0 (the only compression method presently defined for PNG) specifies deflate/ inflate compression with a sliding window of at most 32768 bytes. Deflate compression is an LZ77 derivative used in zip, gzip, pkzip, and related programs. Extensive research has been done supporting its patent-free status. Portable C implementations are freely available. Deflate-compressed datastreams within PNG are stored in the "zlib" format, ...

3.6 JPEG

Cf.

- www.ece.purdue.edu/. . . /index.html
  Ray Wolfgang's short JPEG tutorial.
- www.faqs.org/. . . /index.html
  www.faqs.org/. . . /index.html
  JPEG-FAQ part 1 & 2.
- www.faqs.org/. . . /section-17.html
  Compression-FAQ: What is JPEG?
- www.faqs.org/. . . /section-6.html
  Compression-FAQ: Introduction to JPEG.
- www.ztt.fh-worms.de/. . . /node38.html
  Joachim Schwarz und Guido Sörmann: International Standard ISO/IEC 10918 (JPEG)
- www.faqs.org/. . . /section-6.html [75] Introduction to JPEG
- ftp:/. . . /wallace Gregory K. Wallace: The JPEG Still Picture Compression Standard

Abbreviation stands for "Joint Photographic Experts Group", pronounced as "Jay-peg"!

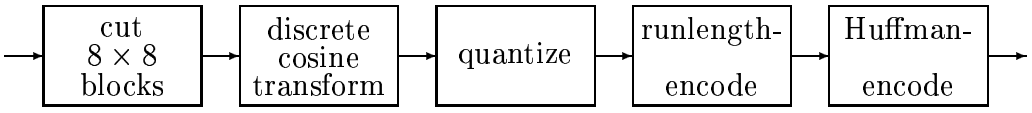
Features:

- High compression rates.
- Colors are coded by 3*8=24 bits.
- Bad at sharp edges, vector-graphics, areas of constant color.
- For a comparison with GIF see
  www.siriusweb.com.html.
  For one with TIFF, GIF and PNG see
  www.cywarp.com/FAQ_TimeCapsules.htm
- Unpacking–Packing degrades the image! With special software it is possible to rotate by 90 degrees and to mirror a JPEG image lossless.

  From www.faqs.org/. . . /section-6.html:
  The official specification of JPEG is not currently available on-line, and is not likely ever to be available for free because of ISO and ITU copyright restrictions. You can order it from your national standards agency as ISO standards

IS 10918-1, 10918-2, 10918-3, or as ITU-T standards T.81, T.83, T.84. See ftp://ftp.uu.net/graphics/jpeg/jpeg.documents.gz for more info.

Encoding in short, see
www.ztt.fh-worms.de/. . . /node44.html
and www.ece.purdue.edu/. . . /index.html:



Jpeg encoding

- Split image in NxN blocks $(A(i,j))_{0\le i,j<N}$, where $N = 8$.
- Apply discrete Cosine Transform to each block, i.e. determine for $0 \le r, s < N = 8$

$$B(r,s) := C_r\, C_s \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} A(i,j)\, \cos\left(\frac{r(2i+1)}{2N}\pi\right)\, \cos\left(\frac{s(2j+1)}{2N}\pi\right).$$

  with $C_0 = \sqrt{\frac{1}{N}}$ and $C_i = \sqrt{\frac{2}{N}}$ for $i > 0$.
- Quantize the matrices B by dividing component-wise with a fixed 8×8 matrix having larger coefficients near bottom-right corner. See
  www.ztt.fh-worms.de/. . . /node37.html
- Binary Encode (zigzag runlength encoding combined with Huffman encoding) the resulting matrices. See
  www.ztt.fh-worms.de/. . . /node49.html
  www.ece.purdue.edu/. . . /jpgcmpr1.html

In more details this means:

3.6.1 YUV

From www.faqs.org/. . . /section-6.html:
1. Transform the image into a suitable color space. This is a no-op for grayscale, but for color images you generally want to transform RGB into a luminance/ chrominance color space (YCbCr, YUV, etc). The luminance component is grayscale and the other two axes are color information. The reason for doing this is that you can afford to lose a lot more information in the chrominance

components than you can in the luminance component: the human eye is not as sensitive to high-frequency chroma info as it is to high-frequency luminance. (See any TV system for precedents.) You don't have to change the color space if you don't want to, since the remainder of the algorithm works on each color component independently, and doesn't care just what the data is. However, compression will be less since you will have to code all the components at luminance quality. Note that colorspace transformation is slightly lossy due to roundoff error, but the amount of error is much smaller than what we typically introduce later on.

2. (Optional) Downsample each component by averaging together groups of pixels. The luminance component is left at full resolution, while the chroma components are often reduced 2:1 horizontally and either 2:1 or 1:1 (no change) vertically. In JPEG-speak these alternatives are usually called 2h2v and 2h1v sampling, but you may also see the terms "411" and "422" sampling. This step immediately reduces the data volume by one-half or one-third. In numerical terms it is highly lossy, but for most images it has almost no impact on perceived quality, because of the eye's poorer resolution for chroma info. Note that downsampling is not applicable to grayscale data; this is one reason color images are more compressible than grayscale.

3.6.2 DCT. Discrete Cosine Transformation

The Discrete Cosine Transformation is a variant of the Fourier-Transform (Fourier-Series). The basic idea behind these transformations is that any (periodic) function can be decomposed in sine and cosine functions with frequencies being multiples of the given periodicity. Consider the symmetric $N \times N$-matrix

$$A_2 := \begin{pmatrix} 1 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & \ddots & & \vdots \\ 0 & -1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & -1 & 0 \\ \vdots & & \ddots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{pmatrix}$$

Its Eigenvectors are $v_k : j \mapsto \cos((j+\frac{1}{2})\frac{k\pi}{N})$ for $0 \le k < N$ with Eigenvalues $\lambda_k := 2\Big(1-\cos\big(\frac{k\pi}{N}\big)\Big)$ for $1 < k < N-1$, since for $\theta := \frac{k\pi}{N}$ we have

$$\cos\Big(\frac{1}{2}\theta\Big) - \cos\Big(\frac{3}{2}\theta\Big) = 2\Big(1-\cos(\theta)\Big)\cos\Big(\frac{1}{2}\theta\Big)$$
$$-\cos\Big((j-\frac{1}{2})\theta\Big) + 2\cos\Big((j+\frac{1}{2})\theta\Big) - \cos\Big((j+\frac{3}{2})\theta\Big) = 2(1-\cos\theta)\cos\Big((j+\frac{1}{2})\theta\Big),$$
$$\cos((N-\frac{1}{2})\theta) - \cos\Big((N-\frac{3}{2})\theta\Big) = 2\Big(1-\cos(\theta)\Big)\cos\Big((N-\frac{1}{2})\theta\Big)$$

using

$$\cos(a)+\cos(b) = 2\cos\Big(\frac{a+b}{2}\Big)\cos\Big(\frac{a-b}{2}\Big).$$

Since A_2 is symmetric, and the sequence of Eigenvalues is strictly monotone increasing, we conclude that the Eigenvectors v_k are orthogonal, thus any vector $a \in \mathbb{R}^N$ can be reconstructed from the projections $\langle a, v_k\rangle$ via

$$a = \sum_{k=0}^{N-1} \frac{\langle a, v_k\rangle}{\|v_k\|^2} v_k$$

The norms are $\|v_0\|^2 = N$ and $\|v_k\|^2 = \frac{N}{2}$ for $k \neq 0$, since

$$\begin{aligned}\|v_k\|^2 &= \sum_{j<N} \cos\Big((j+\frac{1}{2})\theta\Big)^2 = \frac{1}{2}\sum_{j<N}\Big(1+\cos\Big((2j+1)\theta\Big)\Big)\\ &= \frac{N}{2} + \frac{1}{2}\Re\Big(\sum_{j=0}^{N-1} e^{i\theta(2j+1)}\Big) = \frac{N}{2} + \frac{1}{2}\Re\Big(\sum_{j=0}^{2N-1} e^{i\theta j} - \sum_{j=0}^{N-1} e^{i\theta 2j}\Big)\\ &= \frac{N}{2} + \frac{1}{2}\Re\Big(\frac{e^{i\theta 2N}-1}{e^{i\theta}-1} - \frac{e^{i\theta 2N}-1}{e^{i\theta 2}-1}\Big) = \frac{N}{2} + 0,\end{aligned}$$

using $2\cos(a)^2 = 1+\cos(2a)$ and $e^{ia} = \cos(a) + i\sin(a)$. So

$$\mathcal{F}(a)_k := \frac{\langle a, v_k\rangle}{\|v_k\|} = \begin{cases} \frac{1}{\sqrt{N}}\sum_{j<N} a_j & \text{für } k = 0 \\ \sqrt{\frac{2}{N}}\sum_{j<N} a_j \cos\Big((j+\frac{1}{2})\frac{k\pi}{N}\Big) & \text{für } k > 0 \end{cases}$$

defines an ISOMETRY $\mathbb{R}^N \to \mathbb{R}^N$, i.e. is length-preserving with respect to the euclidean norm, since

$$\begin{aligned}\|a\|^2 &= \left(\sum_{k<N} \frac{\langle a, v_k\rangle}{\|v_k\|^2} v_k\right)^2 \\ &= \sum_{k<N} \left(\frac{\langle a, v_k\rangle}{\|v_k\|^2} v_k\right)^2 \\ &= \sum_{k} \left(\frac{\langle a, v_k\rangle}{\|v_k\|}\right)^2 \\ &= \sum_{k} (\mathcal{F}(a)_k)^2.\end{aligned}$$

The 2-dimensional DCT is now defined by applying the 1-dimensional DCT to rows and columns, i.e.

$$\mathcal{F}(a)_{r,s} := C_r\, C_s \sum_i \sum_j a_{i,j} \cos\left(\frac{r(2i+1)}{2N}\pi\right) \cos\left(\frac{s(2j+1)}{2N}\pi\right)$$

Thus with respect to the ∞-norm we get for the 2-dimensional DCT:

$$|\mathcal{F}(a)_{r,s}| \le \|a\|_2 \le \sqrt{\sum_{i,j<N} \|a\|_\infty^2} = \sqrt{N^2}\|a\|_\infty < N * 128 = 1024.$$

Note that here we used a shift for the original data $0 \le a_{i,j} < 256$ to $-128 \le a_{i,j} - 128 < 128$.

See also www.ztt.fh-worms.de/. . . /node34.html

3.6.3 Quantization

www.ztt.fh-worms.de/. . . /node37.html

> From www.ztt.fh-worms.de/. . . /node37.html :
> **Quantisierung**
>
> Sowohl beim JPEG, als auch beim MPEG - Kompressionsalgorithmus, folgt auf die DCT der Vorgang der Quantisierung um den Wertebereich der durch die DCT ermittelten Koeffizienten zu verringern. Die eingesetzten Quantisierungstechniken unterscheiden sich zwar bei der JPEG und MPEG Komprimierung, sind jedoch in ihrem Wesen gleich. Da es hier nur um das Prinzipielle Verständnis

geht, wird an dieser Stelle nur die einfachste Methode, sozusagen der kleinste gemeinsame Nenner beschrieben.

Bei der Quantisierung werden alle, durch die DCT ermittelten, Frequenzwerte durch verschiedene oder gleiche Werte einer zweidimensionalen Quantisierungstabelle geteilt. Eine solche Quantisierungstabelle könnte z.B. wie folgt aussehen:

$$Q = \begin{pmatrix} 8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{pmatrix}$$

Betrachtet man diese Matrix, so fällt auf, daß die hohen Frequenzen durch höhere Werte dividiert werden, als die niedrigeren Frequenzen. Dies liegt daran, daß man davon ausgeht, daß die hohen Frequenzen ein Rauschen repräsentieren und die niedrigen Frequenzen eine Struktur in einem Bild beschreiben. Da das menschliche Wahrnehmungsvermögen strukturorientiert ist, kann die Quantisierung in den Bereichen der hohen Frequenzen größere Quantisierungsstufen ansetzen als in den sensibleren Bereichen der niedrigeren Frequenzen, wo sonst sehr schnell "Blockeffekte" wahrgenommen werden würden.

Probleme entstehen bei diesem Verfahren, falls ein Bild tatsächlich einen hohen Informationsanteil in dem hohen Frequenzbereich enthält, wie z.B. oft in synthetischen Bildern vorkommt, oder wenn sehr kleine Schrift im Bild ist. Hier muß die Quantisierungsmatrix entsprechend angepaßt werden, um einen zu hohen Informations - und Inhaltsverlust des Bildes zu vermeiden.

Der Nutzen, der aus der Anwendung dieses Verfahrens gezogen wird, besteht darin, daß der Wertebereich der Koeffizienten der DCT - Transformation verringert wird. D.h. also, daß aus einer 100 beispielsweise eine 10 wird, und da diese Werte über eine variable Längen - Kodierung abgelegt werden, werden auf diese Art einige Bits oder gar Bytes gespart.

3.6.4 Encoding

From www.ece.purdue.edu/. . . /jpgcmpr1.html :
After quantization, it is not unusual for more than half of the DCT coefficients

to be equal to zero. JPEG incorporates run-length coding to take advantage of this. For each non-zero DCT coefficient, JPEG records the number of zeros that preceded the number, the number of bits needed to represent the number's amplitude, and the amplitude itself. To consolidate the runs of zeros, JPEG processes DCT coefficients in the zigzag pattern shown in figure two:

Encoding

The number of previous zeros and the bits needed for the current number's amplitude form a pair. Each pair has its own code word, assigned through a variable length code (for example Huffman, Shannon-Fano or Arithmetic coding). JPEG outputs the code word of the pair, and then the codeword for the coefficient's amplitude (also from a variable length code). After each block, JPEG writes a unique end-of-block sequence to the output stream, and moves to the next block. When finished with all blocks, JPEG writes the end-of-file marker.

From www.ztt.fh-worms.de/.../node48.html :

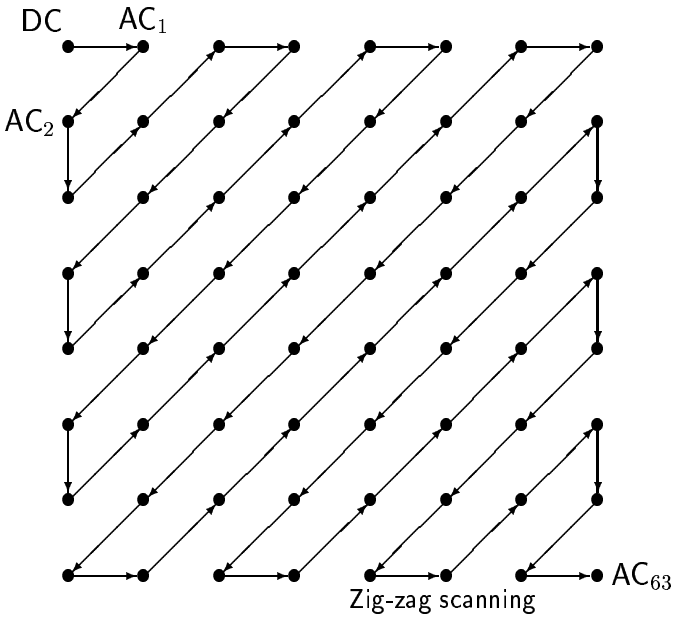
DC Kodierung und Zig-Zag Scanning

Nach der Quantisierung wird der DC-Wert im Gegensatz zu den 63 AC-Werten getrennt behandelt. Die DC-Werte werden stets abhängig vom zuletzt kodierten Block der selben Bildkomponente kodiert. Dabei gilt:

$$DIFF = DC_i - DC_{i-1}$$

Der Wert in DIFF wird als DC-Wert kodiert. Wenn kein Vorgängerblock existiert, zum Beispiel beim Beginn der Kodierung, so wird der DC-Wert kodiert.

Die AC-Werte werden unabhängig von anderen Werten kodiert. Danach werden alle kodierten Werte mit Hilfe des Zig-Zag Scanning eingelesen.



Zig-Zag Scanning

Das Zig-Zag Scanning ist ein Verfahren, um die Werte einer 8x8 Felder großen Matrix in ein eindimensionales 64 Felder großes Array zu übertragen. Es ist ebenfalls möglich, ein eindimensionales Array in ein zweidimensionales umzuwandeln.

www.ztt.fh-worms.de/. . . /node49.html

From www.ztt.fh-worms.de/. . . /node49.html :

Entropie Kodierung

Die Entropie Kodierung ist der letzte Komprimierungsschritt. Hier werden die quantisierten DCT-Werte aufgrund ihrer statistischen Häufigkeit kodiert. Der JPEG Standard umfaßt zwei mögliche Komprimierungsmethoden:

- Huffman-Kodierung
- Arithmetische Kodierung

Bei der Huffman-Kodierung wird gefordert, daß die Anwendung eine oder mehrere Huffmantabellen spezifiziert. Diese Huffmantabellen werden dann zur Kodierung als auch zur Dekodierung verwendet. Das bedeutet, daß neben den Quantisierungstabellen auch die Huffmantabellen mit übertragen werden müssen.

From www.ztt.fh-worms.de/.../node53.html :

Entropie Zwischenkodierung

Mit Hilfe der Entropie Zwischenkodierung werden die AC-Werte so kodiert, daß sie später leicht mit Hilfe des Huffman-Verfahrens komprimiert werden können. Jeder von Null verschiedene AC-Wert wird mit der Lauflänge der vorangehenden AC-Nullwerte dargestellt. Dabei wird im Zig-Zag Verfahren abgetastet. Die AC-Werte sehen dann folgendermaßen aus:

AC-Wert= [Lauflänge|Größe][Amplitude]

Das Symbol-1 stellt zwei Informationen dar: die LAUFLÄNGE und die GRÖSSE. Symbol-2 repräsentiert eine einzige Information: die AMPLITUDE, also der Wert eines von Null verschiedenen AC-Koeffizienten. Die LAUFLÄNGE ist die Anzahl von Nullwerten bei Zig-Zag Abtastung, die vor dem von Null verschiedenen AC-Wert stehen. GRÖSSE ist die Anzahl an Bits die gebraucht wird, um AMPLITUDE binär zu kodieren.

LAUFLÄNGE repräsentiert Nullfolgen der Länge 0 bis 15. Da LAUFLÄNGE aber größer als 15 werden kann wird Symbol-1 mit dem Wert (15,0) als 16 interpretiert. Es kann maximal 3 mal der Ausdruck (15,0) erscheinen. Folgt darauf kein von Null verschiedener AC-Wert mehr, wird das Block-Endezeichen (EOB) Symbol-1 mit dem Wert (0,0) gesetzt. Es markiert das Ende der Data Unit.

Die Darstellungsweise der DC-Werte ist einfacher.

DC-Wert= [Größe][Amplitude]

Symbol-1 repräsentiert hier nur die Anzahl der Bits die gebraucht werden, um Symbol-2, also die AMPLITUDE des DC-Wertes zu kodieren.

From www.ztt.fh-worms.de/.../node54.html :

Variable Längenkodierung

Für die AC- und DC-Koeffizienten wird nun das Symbol-1 mit einem variablen Längen-Code (VLC) kodiert. dieser Code stammt aus der Huffmantabelle, die vom Anwender festgelegt wurde. Jedes Symbol-2 wird dagegen mit einem variablen Längen-Integer-Code (VLI) kodiert, der in der unten stehenden Tabelle dargestellt ist.

SIZE	AMPLITUDE	CODE
1	-1,1	0,1
2	-3,-2,2,3	00,01,10,11
3	-7..-4,4..7	000,..,011,100,..,111
4	-15..-8,8..15	0000,..,0111,1000,..,1111
5	-31..-16,16..31	00000,..,01111,10000,..,11111
6	-63..-32,32..63	000000,..,011111,100000,..,111111
7	-127..-64,64..127	0000000,..,0111111,1000000,..,1111111
8	-255..-128,128..255	00000000,..,01111111,10000000,..,11111111
9	-511..-256,256..511	000000000,..,011111111,100000000,..,111111111
10	-1023..-512,512..1023	0000000000,..,0111111111,1000000000,..,1111111111

VLC und VLI sind beide variable Längencodes, VLI ist aber kein Huffman Code. Ein wichtiger Unterschied zwischen den beiden Codes ist, daß die Länge des Huffman Codes (VLC) bis zur Dekodierung nicht bekannt ist. Die Länge des VLI Codes ist dagegen im vorherigen VLC Code gespeichert.

3.6.5 Example

For an example see: www.ztt.fh-worms.de/. . . /node55.html

3.7 Comparison

See www.cywarp.com/FAQ_TimeCapsules.htm treating the following questions:

- What are suitable file formats to use?
- Should you compress images?
- What is the TIF file format?
- What is PNG file format?
- What is JPG file format?
- What is GIF file format?

Which format should be used for which purpose:

- PNM For temporary fast storage.
- GIF For images with few colors, for the WEB.
- PNG For ray-tracing photo realistic pictures and storing as master file.
- JPEG For photos to be published on the WEB if highest quality is not highest priority but size and transmission time is.

Chapter 4

Euclidean and Projective Geometry

Literature

- 2D-Transformations:
  [FvDFH90, 5.1ff] and [PK87, 4]
- 3D-Transformations:
  [FvDFH90, 5.6ff] and [PK87, 6]

Links

- www.inrialpes.fr/.../isprs96.html
  An Introduction to Projective Geometry (for computer vision).html
  Projective Geometry for Image Analysis: Comprehensive Introduction to projective versus affine space.
- www.martinb.com/.../index.htm
  keywords: Rotation, Euler Angles, NASA Standard Aerospace
- www.martinb.com/.../index.htm
  keywords: Rotation using Quaternions
- www.martinb.com/.../index.htm
  keywords: Conversion of Rotations

- graphics.cs.ucdavis.edu/. . . /Coordinate-Systems.html
  keywords: Handedness
- www.martinb.com/. . . /index.htm
  keywords: Handedness (Nice picture)
- graphics.cs.ucdavis.edu/. . . /Affine-Barycentric-and-Convex.html
  graphics.cs.ucdavis.edu/. . . /Affine-Barycentric-and-Convex.pdf
  keywords: Coordinate System, Barycentric Coordinates
- graphics.cs.ucdavis.edu/. . . /Frames.html
  graphics.cs.ucdavis.edu/. . . /Frames.pdf
  keywords: Frames as Coordinate System of affine space
- moving.htm
  keywords: Transformations: Translation, Scaling, Rotation
- [pov:6.3 POV-Ray Coordinate System]
- [pov:6.3.1 Transformations]
- [pov:3.1.1 Understanding POV-Ray's Coordinate System:]
- [pov:Types of Projection]
- **Pov-Ray** tutorial
  www.f-lohmueller.de/. . . /povtuto0.htm
- Elementary Transformations
  www.f-lohmueller.de/. . . /basics1e.htm#start
- almond.srv.cs.cmu.edu/. . . /transform_2.pdf
  keywords: Slides on Transformations
- almond.srv.cs.cmu.edu/. . . /04-transform.pdf
  keywords: 6-Slides on Transformations
  almond.srv.cs.cmu.edu/. . . /04-transform.pdf
  keywords: 2-Slides on Transformations
  almond.srv.cs.cmu.edu/. . . /04-transform.pdf
  keywords: 1-Color-Slide on Transformations
- almond.srv.cs.cmu.edu/. . . /05-viewing.pdf
  keywords: 6-Slides on Viewing-Projections
  almond.srv.cs.cmu.edu/. . . /05-viewing.pdf
  keywords: 2-Slides on Viewing-Projections

almond.srv.cs.cmu.edu/. . . /05-viewing.pdf
keywords: 1-Color-Slide on Viewing-Projections

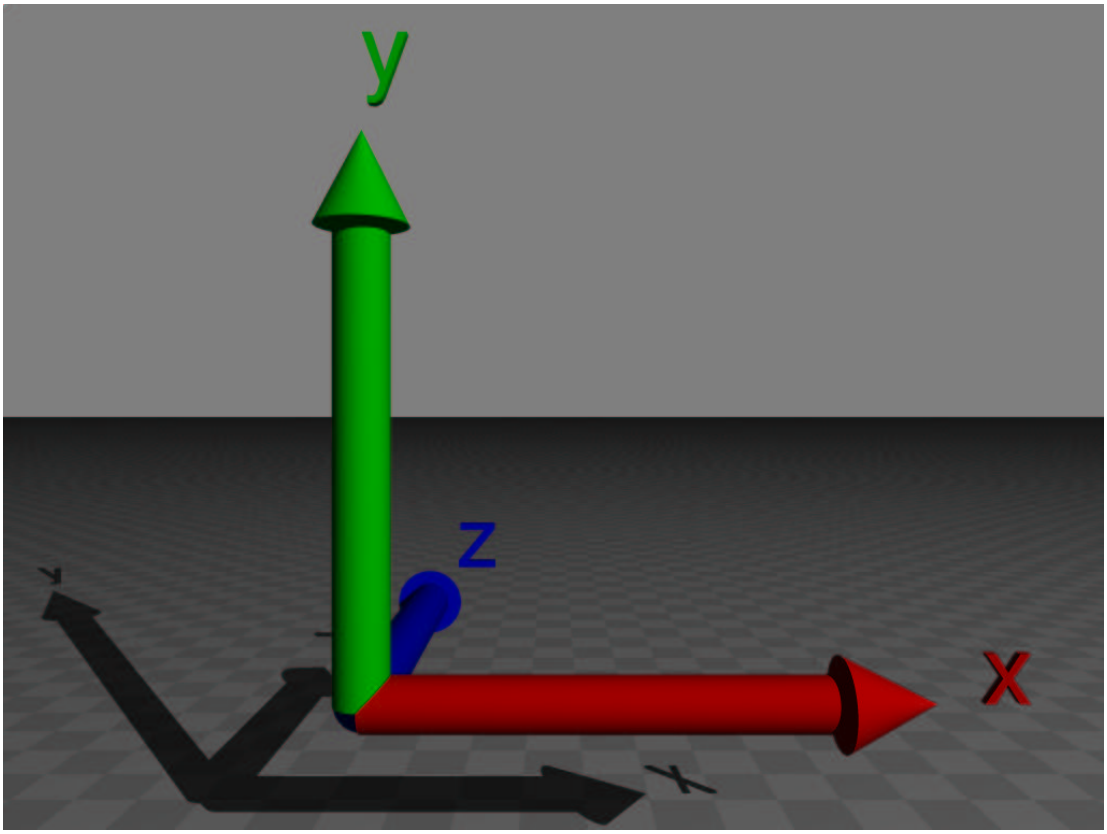
- graphics.cs.ucdavis.edu/. . . /Camera-Transform.pdf
  keywords: Camera Projection

4.1 Affine space

The objects we want to display are situated in the space surrounding us. We will neglect that this space is curved by gravitation according to Einsteins general relativity theory (or 26 dimensional according to some quantum theoretic approaches) and for sake of simplicity we may assume that this is an affine 3-dimensional space. So after choosing some reference frame, i.e. a zero-point and three independent vectors we may identify this space with the standard 3-dimensional vector space $\mathbb{R}^3$. Its points are uniquely given by 3 real coordinates (a, b, c) denoted by $\langle a, b, c\rangle$ in **Pov-Ray**. In **Pov-Ray** one may use as shortcut for a vector with equal coordinates the real number given by this coordinate, e.g. $0 = \langle 0, 0, 0\rangle$. The standard basis with respect to the given frame is denoted

$$x := \langle 1, 0, 0\rangle, \qquad y := \langle 0, 1, 0\rangle, \qquad z := \langle 0, 0, 1\rangle$$

in **Pov-Ray**, and are interpreted as pointing right, up and forward. This is a let-handed system, i.e. take your left hand, let the thumb point in direction x and the index finger point in direction y, then the middle finger points in direction z.

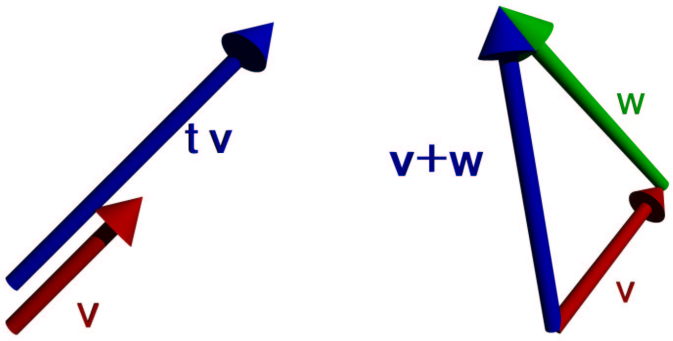


Pov-Ray's left-handed coordinate system

Note that the second basic vector y and not z is pointing up. The idea behind this choice ist that usually we will project onto some vertical plane, so the first two coordinates should be associated to this plane.

On $\mathbb{R}^3$ we have the usual vector operations:

$$\lambda \cdot \langle a_1, b_1, c_1 \rangle = \langle \lambda \cdot a_1, \lambda \cdot b_1, \lambda \cdot c_1 \rangle \qquad \text{addition}$$
$$\langle a_1, b_1, c_1 \rangle + \langle a_2, b_2, c_2 \rangle = \langle a_1 + a_2, b_1 + b_2, c_1 + c_2 \rangle \qquad \text{addition}$$



Scalar multiplication and vector addition

We will also make use of the inner product $\langle v|w\rangle$ (denoted $\mathrm{vdot}(v,w)$ in **Pov-Ray**) of two vector v and w, measuring essentially the angle $\angle(v,w)$ between the two vectors via

$$\cos(\angle(v,w)) = \frac{\mathrm{vdot}(v,w)}{\sqrt{\mathrm{vdot}(v,v)}\sqrt{\mathrm{vdot}(w,w)}}$$

and the length

$$\mathrm{vlength}(v) := \|v\| := \sqrt{\mathrm{vdot}(v,v)}.$$

Thus $\mathrm{vdot}(v,w)$ is the length of v multiplied with the length of the normal projection of w onto v. Using the length we may normalize any vector $v \neq 0$ to obtain a vector of length 1 pointing in the same direction as v:

$$\mathrm{vnormalize}(v) := \frac{1}{\mathrm{vlength}(v)}\, v.$$

Furthermore we have the cross-product (denoted $\mathrm{vcross}(v,w)$ in **Pov-Ray**) of two vectors, which gives a vector normal to v and w of length the area of the parallelogram generated by v and w and such that $v, w, \mathrm{vcross}(v,w)$ is positively oriented. So we have an euclidean space.

4.2 Transformations

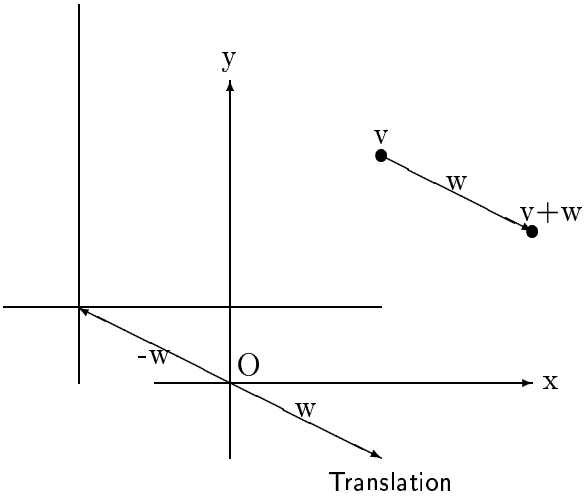
We want to transform the objects (and thus the points) in our space around.

4.2.1 Translations

The translation T_w in direction of a vector w is given in coordinates by

$$T_w : v \mapsto v + w$$

In **Pov-Ray** this operation is described as `translate w`. Note that this can be also considered as changing the reference frame, by replacing 0 by $-w$.



4.2.2 Scaling

We can scale the vectors v with a constant factor w, i.e. $v \mapsto w \cdot v$. In case $w = -1$ we have a point reflection. We can even scale with a vector w by multiplying the corresponding coordinates by that of w, i.e. $v \mapsto \langle w_1\, v_1, w_2\, v_2, w_3\, v_3 \rangle$.

Syntax in **Pov-Ray**: `scale w`

In case all coordinates of w are 1 except one which is -1 this is a reflection on the plane given by the set of points where the corresponding coordinate is 0. So for $w = \langle -1, 1, 1 \rangle$ we get a reflection on the y-z-plane.

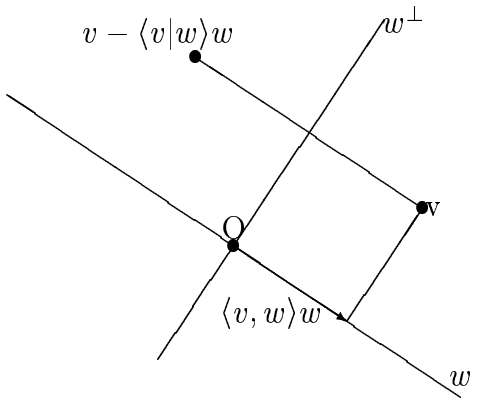
4.2.3 Reflections/Mirroring

Now we want to describe the reflection S_w on a plane orthogonal to some (unit) vector w. Since any vector v can be written as

$$v = \langle v|w \rangle w + (v - \langle v|w \rangle w),$$

where $v - \langle v|w \rangle w$ is orthogonal to w (use $\langle v - \langle v|w \rangle w, w \rangle = \langle v|w \rangle (1 - \|w\|^2) = 0$), we get

$$\begin{aligned} S_w(v) &= S_w\Big(\langle v|w \rangle w\Big) + S_w\Big(v - \langle v|w \rangle w\Big) \\ &= -\langle v|w \rangle w + \Big(v - \langle v|w \rangle w\Big) = v - 2\,\langle v|w \rangle w. \end{aligned}$$



Reflection

In coordinates:

$$\begin{aligned}(v_1, v_2, v_3) &\mapsto \Big(v_1 - 2\Big(\sum_k v_k\, w_k\Big)\, w_1, v_2 - 2\Big(\sum_k v_k\, w_k\Big)\, w_2, v_3 - 2\Big(\sum_k v_k\, w_k\Big)\, w_3\Big)\\ &= \begin{pmatrix} 1-2w_1^2 & -2w_1\, w_2 & -2w_1\, w_3 \\ -2w_2\, w_1 & 1-2w_2^2 & -2w_2\, w_3 \\ -2w_3\, w_1 & -2w_3\, w_2 & 1-2w_3^2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}\\ &= (\mathrm{id} - 2\, w \cdot w^t) \cdot v = v - 2\, w\, \langle w|v\rangle\end{aligned}$$

These reflections are length preserving:

$$\begin{aligned}(\mathrm{id} - 2\, w \cdot w^t)^t \cdot (\mathrm{id} - 2\, w \cdot w^t) &= (\mathrm{id} - 2\, w \cdot w^t) \cdot (\mathrm{id} - 2\, w \cdot w^t)\\ &= \mathrm{id} - 2\, w \cdot w^t - 2\, w \cdot w^t + 4\, w \cdot w^t \cdot w \cdot w^t = \mathrm{id},\end{aligned}$$

or, because of Pythagoras,

$$\begin{aligned}\|v - 2\,\langle v|w\rangle w\|^2 &= \|v - \langle v|w\rangle w\|^2 + \|\pm\langle v, w\rangle w\|^2\\ &= \|v - \langle v|w\rangle w + \langle v|w\rangle w\|^2 = \|v\|^2.\end{aligned}$$

Note here, that a linear mapping given by multiplication with the matrix A is length preserving, iff

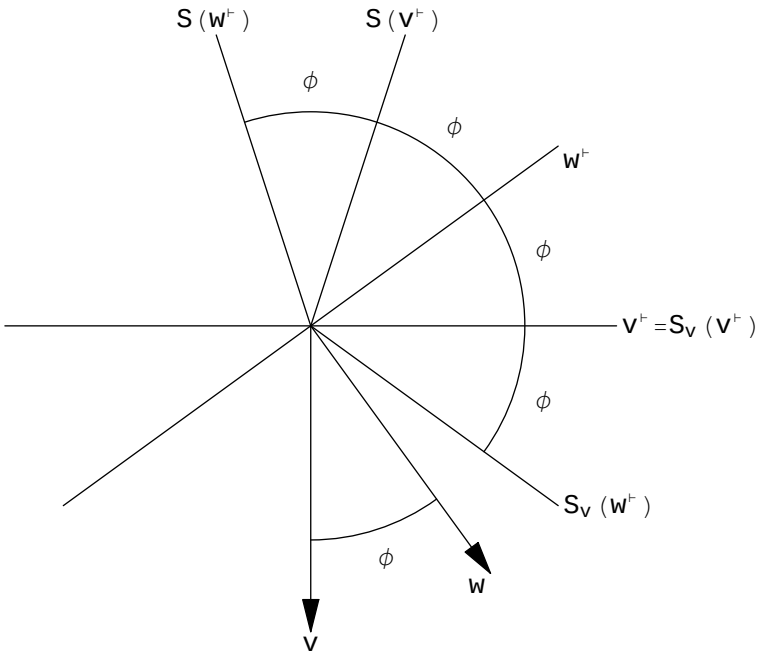
$$v^t \cdot w = \langle v|w\rangle = \langle Av|Aw\rangle = (Av)^t \cdot Aw = v^t\, A^t\, A\, w$$

for all v and w, i.e. iff $A^t \cdot A = \mathrm{id}$.

The composition of the two 2-dimensional reflections $\langle a, b\rangle \mapsto \langle -a, b\rangle$ and $\langle a, b\rangle \mapsto \langle a, -b\rangle$ amounts to mirroring $v \mapsto -v$ at the center O. The composition of two general reflections is a rotation:

$$\begin{aligned}(S_{w_2} \circ S_{w_1})(v) &= S_{w_1}(v) - \langle S_{w_1}(v)|w_2\rangle w_2\\ &= v - \langle v|w_1\rangle w_1 - \langle v - \langle v|w_1\rangle w_1|w_2\rangle w_2\\ &= v - \langle v|w_1\rangle w_1 - \langle v|w_2\rangle w_2 + \langle v|w_1\rangle\langle w_1|w_2\rangle w_2\end{aligned}$$

This is a rotation around $w_1 \times w_2$ by twice the angle between w_1 und w_2.



Rotation as two-fold reflection

4.2.4 Rotations and Euclidean Motions

A general reflection at the plane $w^\perp$ can be described by first rotating w into some vector v, then reflect at $v^\perp$ and now rotate back v to w. In fact, let $u := v + w$. Then this first rotation is given by $S_u \circ S_w = R = S_v \circ S_u$ and we have

$$S_v \circ R = S_v \circ (S_v \circ S_u) = (S_v)^2 \circ S_u = S_u = S_u \circ (S_w)^2 = R \circ S_w.$$

How can rotations be described by matrices? Let us first consider 2-dimensional space. The rotation by 90^o maps any vector v to a normal vector $v^\perp$ of the same length. There are only two possibilities for $\langle a, b\rangle^\perp$ namely $\pm\langle -b, a\rangle$, and the one with $+$ is rotation in the positive direction, i.e. counterclockwise. The matrix corresponding to this rotation is given by

$$R_{\pi/2} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

Now what about a rotation around 0 by an arbitrary angle φ? By elementary geometric calculations we see, that the image of x is $\cos\varphi\, x + \sin\varphi\, y$, and the

image of y which is x turned 90^o counterclockwise is the vector $-\sin\varphi\, x + \cos\varphi\, y$. Thus in coordinates this linear mapping is given by the unitarian matrix

$$\begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}$$

Note that this can also be paraphrased by "the reference frame is rotated in the opposite direction by φ". In three dimensions a rotation by φ around the z-axes does not change the z-coordinate and rotates the (x, y)-coordinates as before and thus is given by the unitarian matrix

$$\begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In **Pov-Ray**: `rotate` φ`*z`.

Similarly rotations around the x- and around the y-axes are given by

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} \text{ and } \begin{pmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{pmatrix}$$

In **Pov-Ray**: `rotate` φ`*x` and `rotate` φ`*y`.

Note that the composition of two rotations by angles φ and ψ around the same center is the rotation by angle $\varphi + \psi$. Expressing this via the corresponding matrices gives the addition laws for sin and for cos.

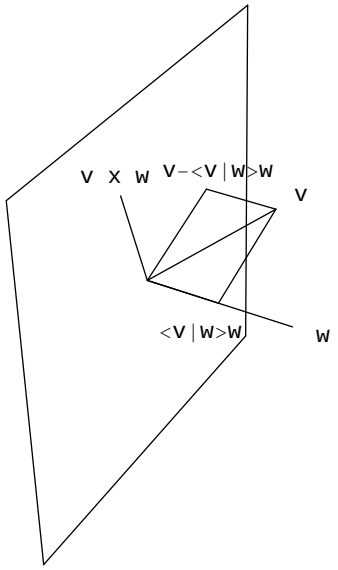
All these rotation matrices M (as well as arbitrary compositions of such) satisfy $\det(M) = 1$ and $M^t \cdot M = \mathrm{id}$, thus are special orthogonal matrices.

A general rotation around the axis spanned by the unit vector w by the angle φ is given by considering the orthogonal frame given by w, $v \times w$, $v - \langle v|w\rangle w$. The length of these vectors are 1, $\sin(\angle(v, w))\|v\|$, $\sin(\angle(v, w))\|v\|$. The vector v is given in this frame as

$$v = \langle v|w\rangle \cdot w + 1 \cdot (v - \langle v|w\rangle w) + 0 \cdot v \times w,$$

so the rotation is

$$v \mapsto \langle v|w\rangle \cdot w + \cos(\varphi) \cdot (v - \langle v|w\rangle w) + \sin(\varphi) \cdot v \times w$$



Orthogonal decomposition of a vector v

Note that

$$w \times (v \times u) = (u \times v) \times w = \langle u|w\rangle v - \langle v|w\rangle u$$
$$\text{and hence} \qquad v \times (v \times u) = \langle u|v\rangle v - \langle v|v\rangle u = \langle u|v\rangle v - u.$$

See www.martinb.com/. . . /index.htm.

Let f be an arbitrary length preserving mapping (a so called EUCLIDEAN MOTION), i.e. $\|f(v_1) - f(v_2)\| = \|v_1 - v_2\|$ for all v_1, v_2. Up to the translation by $f(O)$ (i.e. replacing f by $v \mapsto f(v) - f(0)$) it preserves also the origin O and hence $\|f(v)\| = \|v\|$ for all v. Furthermore by the polarization equality

$$2\langle v|w\rangle = \|v\|^2 + \|w\|^2 - \|v - w\|^2$$

we get

$$\begin{aligned} 2\langle f(v)|f(w)\rangle &= \|f(v)\|^2 + \|f(w)\|^2 - \|f(v) - f(w)\|^2 \\ &= \|v\|^2 + \|w\|^2 - \|v - w\|^2 = 2\langle v|w\rangle, \end{aligned}$$

i.e. f preserves the inner product and hence maps the standard orthogonal basis $e_1 = x$, $e_2 = y$ and $e_3 = z$ to some other orthonormal basis $f_1 := f(e_1)$, $f_2 := f(e_2)$ and $f_3 := f(e_3)$. For a general vector v we have $v = \sum_k v_k\, e_k$ with $v_k = \langle v|e_k\rangle = \langle f(v)|f(e_k)\rangle = \langle f(v)|f_k\rangle$ and hence

$$f(v) = \sum_k \langle f(v)|f_k\rangle f_k = \sum_k v_k\, f_k,$$

i.e. f is linear and, moreover, for the matrix $[f]$ associated to f we have $[f]^t \cdot [f] = \mathrm{id}$, i.e. f is given by an orthogonal matrix: In fact the jk^{th} entry in the matrix $[f]^t \cdot [f]$ is given by

$$\begin{aligned}([f]^t \cdot [f])_{j,k} = [f^* \circ f]_{j,k} = \langle (f^* \circ f)e_j | e_k \rangle = \langle fe_j | fe_k \rangle \\ = \langle e_j | e_k \rangle = \begin{cases} 1 & \text{for} j = k, \\ 0 & \text{otherwise.} \end{cases}\end{aligned}$$

Let now conversely an orthogonal 3×3-matrix A be given. Then $1 = \det(\mathrm{id}) = \det(A^t \cdot A) = \det(A^t) \cdot \det(A) = \det(A)^2$, i.e. $\det(A) \in \{-1, +1\}$. Let us assume first that $\det(A) = 1$.

Let $F := \mathrm{Fix}(A) := \{x \in E : Ax = x\}$ be the set of its fixed points, i.e. the eigenspace for the eigenvalue 1. This is a linear subspace.

We show next, that $\dim(F) > 0$. Let $p(\lambda) := \det(\lambda - A)$ be the characteristic polynomial of A. We have

$$\begin{aligned}p(\lambda) &= \det(\lambda - A) = \det((\lambda - A)^t) = \det(\lambda - A^t) = \det(\lambda - A^{-1}) \\ &= \det(-\lambda \left(\frac{1}{\lambda} - A\right) A^{-1}) = \det(-\lambda) \cdot \det\left(\frac{1}{\lambda} - A\right) \cdot \det A^{-1} \\ &= (-\lambda)^3 p(\frac{1}{\lambda}),\end{aligned}$$

thus we get for $\lambda = 1$ that $2p(1) = 0$. Hence 1 is an eigenvalue.

In case its dimension is 3, we have $A = \mathrm{id}$.

In case its dimension is 2, we find a unit vector w such that $F = w^\perp$. Since F is invariant under A and A is orthogonal the same is true for $F^\perp$. Since $w \in F^\perp$ and A is an isometry we have $A(w) = -w$ and hence A is the reflection at the plane $F = w^\perp$. In fact, $x - \langle x|v\rangle v \in F$ for all $x \in E$, since

$$\langle x - \langle x|v\rangle v | v\rangle = \langle x|v\rangle - \langle x|v\rangle \cdot \langle v|v\rangle = 0.$$

Thus

$$\begin{aligned}Ax = A\Big((x - \langle x|v\rangle v) + \langle x|v\rangle v\Big) = x - \langle x|v\rangle v + \langle x|v\rangle(-v) \\ = x - 2\langle x|v\rangle v.\end{aligned}$$

Remains to consider the case, where $\dim(F) = 1$. Then there exists a unit vector w which spans F. The orthogonal plane $w^\perp = F^\perp$ is also A invariant, so $A|_{F^\perp}$ is orthogonal on this plane and hence a reflection on a line in this plane or a rotation

in this plane. Thus A is a reflection at the plane spanned by w and the reflection line of $A|_{F^\perp}$ or a rotation with axis w.

Therefore any orthogonal mapping on $\mathbb{R}^3$ is the composite of (at most three) reflections. It is a rotation around some axes $v \neq 0$ by some angle φ iff it is a composite of two reflections.
In **Pov-Ray**: $v \mapsto$`vaxis_rotate`(v, w, φ).
Finally the euclidean motions are exactly of the form $v \mapsto R \cdot v + w$, where R is a rotation which is followed by the translation $v \mapsto v + w$.

We show next that any rotation (i.e. special orthogonal matrix) can be obtained by composing 3 of the special rotations discussed above by the so called EULER ANGLES. Consider an airplane or an hang-glider: We have the basis given by the axes of airplane: the direction from the left to the right wing, the vertical direction, and the direction from back to front.

- First we turn the head in the required direction via a rotation R_1 around y by angle ψ called heading.
- Next we raise/lower the head by a rotation R_2 around the new x by an angle θ called attitude.
- Finally we tilt the airplane by a rotation R_3 around the new z by the angle φ called bank.

How do the corresponding matrices look like?

- Clearly
$$[R_1] := \begin{pmatrix} \cos(\psi) & 0 & -\sin(\psi) \\ 0 & 1 & 0 \\ \sin(\psi) & 0 & \cos(\psi) \end{pmatrix}$$
- In the new basis $\mathcal{B}' = (x', y, z')$ we have
$$[R_2]_{\mathcal{B}',\mathcal{B}'} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$
In linear algebra (see [Kri02, 10.28] and [Kri02, 10.20a]) one shows that for the matrix-representation with respect to the original basis $\mathcal{B} = (x, y, z)$ we have
$$\begin{aligned} [R_1] := [R_1]_{\mathcal{B},\mathcal{B}} &= [R_1]_{\mathcal{B}',\mathcal{B}'} = [\mathrm{id}]_{\mathcal{B}',\mathcal{B}} \\ [R_2]_{\mathcal{B},\mathcal{B}} &= [\mathrm{id}]_{\mathcal{B}',\mathcal{B}}\,[R_2]_{\mathcal{B}',\mathcal{B}'}\,[\mathrm{id}]_{\mathcal{B},\mathcal{B}'} \\ &= [R_1]\,[R_2]_{\mathcal{B}',\mathcal{B}'}\,[R_1]^{-1} \end{aligned}$$

The composition $R_2 \circ R_1$ has thus the following form

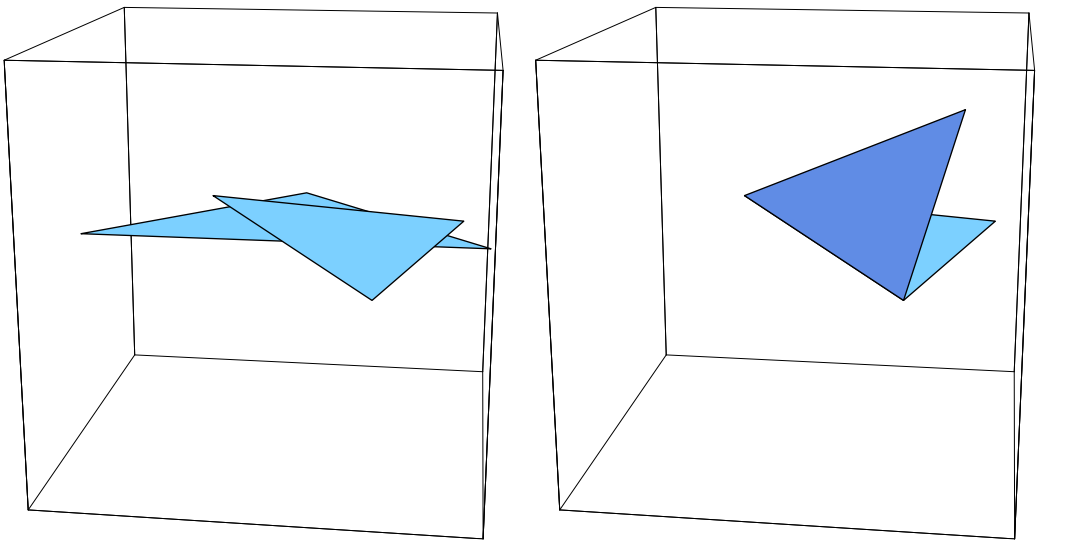
$$[R_2 \circ R_1] = [R_2]\,[R_1] = [R_1]\,[R_2]_{\mathcal{B}',\mathcal{B}'}$$
$$= \begin{pmatrix} \cos(\psi) & 0 & -\sin(\psi) \\ 0 & 1 & 0 \\ \sin(\psi) & 0 & \cos(\psi) \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

- Similarly

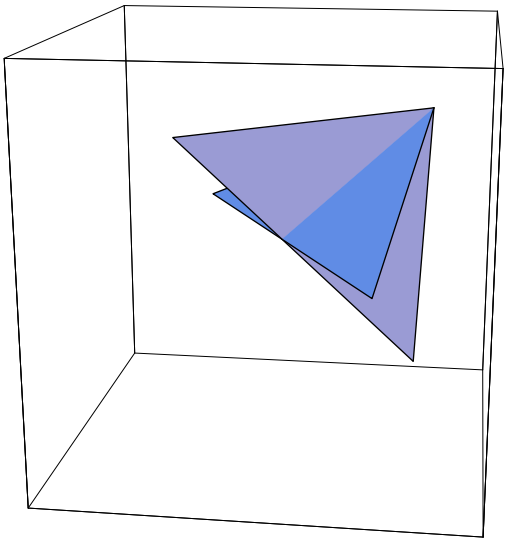
$$[R_3]_{\mathcal{B}'',\mathcal{B}''} = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and hence

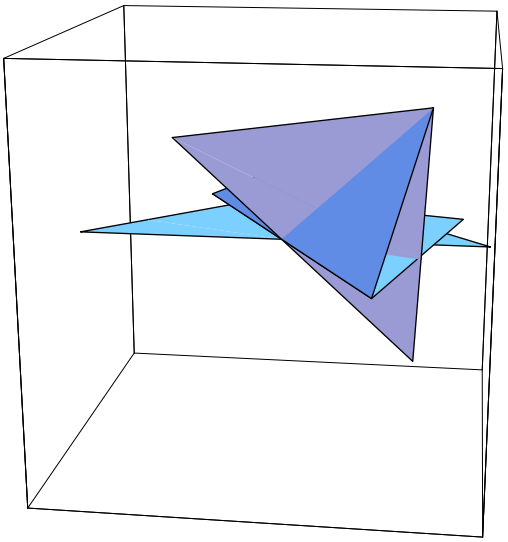
$$[R_3 \circ R_2 \circ R_1] = [R_3]\,[R_2 \circ R_1]$$
$$= [R_2 \circ R_1]\,[R_3]_{\mathcal{B}'',\mathcal{B}''}\,[R_2 \circ R_1]^{-1}\,[R_2 \circ R_1]$$
$$= [R_1]_{\mathcal{B},\mathcal{B}}\,[R_2]_{\mathcal{B}',\mathcal{B}'}\,[R_3]_{\mathcal{B}'',\mathcal{B}''}$$
$$= \begin{pmatrix} \cos(\psi) & 0 & -\sin(\psi) \\ 0 & 1 & 0 \\ \sin(\psi) & 0 & \cos(\psi) \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} \cos(\varphi)\cos(\psi) + \sin(\theta)\sin(\varphi)\sin(\psi) & -\cos(\psi)\sin(\varphi) + \cos(\varphi)\sin(\theta)\sin(\psi) & \cos(\theta)\sin(\psi)) \\ \cos(\theta)\sin(\varphi) & \cos(\theta)\cos(\varphi) & -\sin(\theta) \\ \cos(\psi)\sin(\theta)\sin(\varphi) + \cos(\varphi)\sin(\psi) & \cos(\varphi)\cos(\psi)\sin(\theta) - \sin(\varphi)\sin(\psi) & \cos(\theta)\cos(\psi) \end{pmatrix}$$



R_1 R_2



R_3

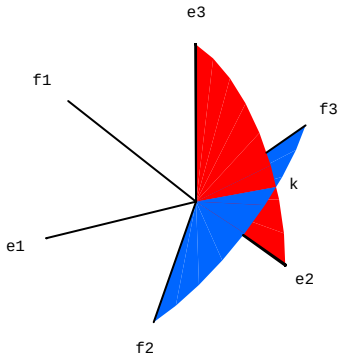


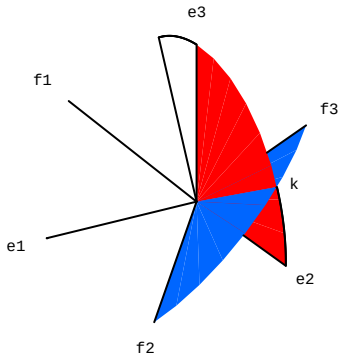
R_1, R_2, R_3

Another decomposition into 3 rotations is given via the following Euler-angles: Let R be a rotation and $f_j := R(e_j)$ be the images of the standard-basis. We would like to express R as composition of 3 rotations around some coordinate-axes. It suffices to describe the images of these rotations on the first 2 vectors e_1 and e_2, since $e_3 = e_1 \times e_2$ is the uniquely determined unit vector normal to e_1 and e_2 such that (e_1, e_2, e_3) is left oriented.

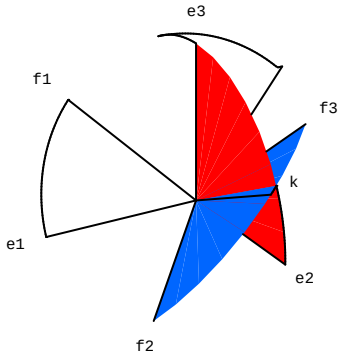
In order to rotate e_1 to f_1 we have to keep an axis $k \in \{e_1\}^\perp \cap \{f_1\}^\perp = \langle\{e_2, e_3\}\rangle \cap \langle\{f_2, f_3\}\rangle$ fixed. In order to rotate afterwards e_2 to f_2 without destroying the assignment $e_1 \mapsto f_1$, we could first rotate e_2 to k around e_1 and at the end rotate k to f_2 around f_1.

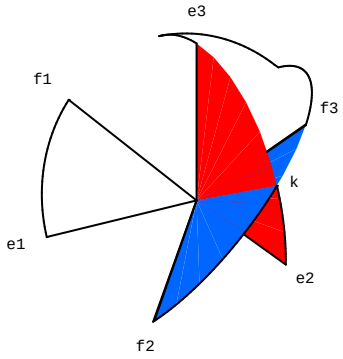
$$\begin{array}{ccccccc} e_1 & \mapsto & e_1 & \mapsto & f_1 & \mapsto & f_1 \\ e_2 & \mapsto & k & \mapsto & k & \mapsto & f_2 \end{array}$$





Fixed axes k R_1





R_2 R_3

Let $\varphi_1, \varphi_2, \varphi_3$ be the so-called Euler-angles of R, given by

$$\varphi_1 := \measuredangle e_2 k; \quad \varphi_2 := \measuredangle e_1 f_1; \quad \varphi_3 := \measuredangle k f_2.$$

Thus the matrix-representation of the corresponding rotations R_1, R_2 and R_3 are

given by:

$$[R_1]_{e_1,e_2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi_1 & -\sin\varphi_1 \\ 0 & \sin\varphi_1 & \cos\varphi_1 \end{pmatrix}$$
$$[R_2]_{e_1,k} = \begin{pmatrix} \cos\varphi_2 & 0 & -\sin\varphi_2 \\ 0 & 1 & 0 \\ \sin\varphi_2 & 0 & \cos\varphi_2 \end{pmatrix}$$
$$[R_1]_{f_1,k} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi_3 & -\sin\varphi_3 \\ 0 & \sin\varphi_3 & \cos\varphi_3 \end{pmatrix}$$

Hence

$$[R_3 \circ R_2 \circ R_1]_{e_1,e_2,e_3} =$$
$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi_1 & -\sin\varphi_1 \\ 0 & \sin\varphi_1 & \cos\varphi_1 \end{pmatrix} \cdot \begin{pmatrix} \cos\varphi_2 & 0 & -\sin\varphi_2 \\ 0 & 1 & 0 \\ \sin\varphi_2 & 0 & \cos\varphi_2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi_3 & -\sin\varphi_3 \\ 0 & \sin\varphi_3 & \cos\varphi_3 \end{pmatrix}$$

4.2.5 Angle-Preserving Mappings

The euclidean motions we have just described as $v \mapsto R \cdot v + w$, where R is a rotation or reflection and w is a translation vector, are length and angle preserving. Now we try to identify those mappings which are only angle preserving.

Lemma (Linear conformal mappings).
Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be linear. Then the following statements are equivalent:

1. *f respects angles, i.e. is* CONFORMAL*;*
2. *$\exists \lambda > 0 : \langle f(x), f(y)\rangle = \lambda \langle x, y\rangle$ for all $x, y \in \mathbb{R}^n$;*
3. *$\exists \mu > 0$: μf is an isometry.*

Proof. $(2 \Leftrightarrow 3)$ is obvious with $\lambda\,\mu^2 = 1$.

$(1 \Leftarrow 2)$ let α be the angle between x and y and α' the one between $f(x)$ und $f(y)$. Then

$$\cos\alpha' = \frac{\langle f(x)|f(y)\rangle}{\|f(x)\| \cdot \|f(y)\|} = \frac{\lambda\langle x|y\rangle}{\sqrt{\lambda}\|x\|\sqrt{\lambda}\|y\|} = \cos\alpha.$$

Thus $\alpha = \alpha'$, and f respects angles.

$(1 \Rightarrow 2)$ We define $\lambda(v) \geq 0$ implicitly by $\langle f(v)|f(v)\rangle =: \lambda(v)\langle v|v\rangle$.

Let v, w be orthonormal vectors, then $(v+w) \perp (v-w)$. Since f is conform, we have:

$$0 = \langle f(v+w)|f(v-w)\rangle = \langle f(v)|f(v)\rangle - \langle f(w)|f(w)\rangle = \lambda(v) - \lambda(w).$$

Thus λ constant on the orthonormal basis $(e_1, \ldots, e_n)$ von $\mathbb{R}^n$. We put $\lambda := \lambda(e_1) = \ldots = \lambda(e_n)$. For an arbitrary vector $v \in \mathbb{R}^n$ we have:

$$\begin{aligned}
& v = \sum_{i=1}^{n} v^i e_i \\
\Rightarrow \quad & \lambda(v) \sum (v^i)^2 = \lambda(v) \left\langle \sum v^i e_i | \sum v^i e_i \right\rangle = \\
& = \left\langle f\left(\sum_i v^i e_i\right), f\left(\sum_j v^j e_j\right)\right\rangle \\
& = \sum_{i,j} v^i v^j \underbrace{\langle f(e_i)|f(e_j)\rangle}_{\lambda\delta_{i,j}} = \lambda \sum_i (v^i)^2 \\
\Rightarrow \quad & \lambda(v) = \lambda \quad \forall v \in \mathbb{R}^n.
\end{aligned}$$

The lemma now follows from:

$$\begin{aligned}
2\langle v|w\rangle &= \langle v+w|v+w\rangle - \langle v|v\rangle - \langle w|w\rangle \\
2\langle f(v)|f(w)\rangle &= \langle f(v)+f(w)|f(v)+f(w)\rangle - \langle f(v)|f(v)\rangle - \langle f(w)|f(w)\rangle \\
&= \lambda\langle v+w|v+w\rangle - \lambda\langle v|v\rangle - \lambda\langle w|w\rangle \\
&= 2\lambda\langle v|w\rangle. \qquad \square
\end{aligned}$$

4.2.6 General Transformations

Beside euclidean motions, conformal linear mappings we have discussed also scaling, where we shrink or stretch the coordinates according to some factors. Scaling can be described by diagonal matrices with the proportionality factors as eigenvalues.

Another kind of mapping of interest is shearing:

$$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix}$$

This is volume preserving, since its determinant is 1.

We would like to have a common formula for all these transformations. For this we embed $\mathbb{R}^3$ into $\mathbb{R}^4$ via $\langle a, b, c\rangle \mapsto \langle a, b, c, 1\rangle$. Then a translation by the vector $v = (v_1, v_2, v_3)$ can be written as

$$\begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} a + v_1 \\ b + v_2 \\ c + v_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & v_1 \\ 0 & 1 & 0 & v_2 \\ 0 & 0 & 1 & v_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix}$$

Any of the other transformations given by 3×3-matrices can be extended to 4×4-matrices via

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

So all compositions are of the same form

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} = v_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} = v_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} = v_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This is written in **Pov-Ray** as:

```
matrix a11, a21, a31,
       a12, a22, a32,
       a13, a23, a33,
       a14, a24, a34
```

Note that this is transposed relative to the mathematical version.

An advantage of this description is to be able to describe perspective (no-linear) projections in the same way, see (4.3.2).

4.3 Projections

- Projections (orthogonal, projective) cf. [FvDFH90, 6] [PK87, 7]
- Projections (orthogonal, projective)
  cf. [FvDFH90, 6] Perspective Projection, Parallel Projection,
  [PK87, 7] Perspective and Parallel Projection.

We have to make 2D-images of a 3D-world. For this we need some projection $\mathbb{R}^3 \to \mathbb{R}^2$.

4.3.1 Orthographic Projection

This is the simplest type of projection. It's most simplest case is 'forgetting the z-coordinate':

$$\langle v_1, v_2, v_3 \rangle \mapsto \langle v_1, v_2 \rangle$$

More general we could project parallel to some (unit) vector w onto the plane $F = w^\perp$ orthogonal to w. This projection is thus given by $v \mapsto v - \langle v|w\rangle w$. In order to express this in coordinates we need some basis of F which is usually called 'up' and 'right'.

An example for a camera statement in **Pov-Ray** is

```
#declare w = <1,2,3>
camera {
  orthographic
  lock_at w
  up <0,3,0>
  right <-4,0,0>
}
```

See [pov:180.html#target_713]

Let's start with a few general remarks about projections.

Lemma.
Let $P \in L(H)$ IDEMPOTENT, *i.e. $P^2 = P$, with other words P is a* PROJECTION. *Then:*

1. *$1 - P$ is also idempotent.*
2. *$\operatorname{Bild} P = \operatorname{Kern}(1 - P)$ and $\operatorname{Kern} P = \operatorname{Bild}(1 - P)$.*
3. *$H = \operatorname{Bild} P \oplus \operatorname{Kern} P$.*

Proof. 1 We get $(1 - P)^2 = 1 - 2\,P + P^2 = 1 - 2\,P + P = 1 - P$.

2 We have $h \in \operatorname{Bild} P \Leftrightarrow h = Pk$ für ein $k \in H \Leftrightarrow Ph = P^2k = Pk = h \Leftrightarrow h \in \operatorname{Kern}(1 - P)$. Furthermore, $\operatorname{Kern} P = \operatorname{Bild}(1 - P)$, since $1 - P$ is idempotent.

3 We have $\operatorname{Bild} P \cap \operatorname{Kern} P = \{0\}$, since $h \in \operatorname{Bild} P$ implies that $Ph = h$ and on the other hand $Ph = 0$ for $h \in \operatorname{Kern} P$. Each $h \in H$ can be written as $h = Ph + (1 - P)h$, with $Ph \in \operatorname{Bild} P$ and $(1 - P)h \in \operatorname{Bild}(1 - P) = \operatorname{Kern} P$. □

Lemma.
For an idempotent $P \in L(H)$ the following statements are equivalent:

4. *P is an orthogonal projection, i.e.* $\operatorname{Kern} P = (\operatorname{Bild} P)^\perp$*;*
5. $\operatorname{Kern} P \perp \operatorname{Bild} P$*;*
6. $\|P\| \leq 1$*, i.e. P is a contraction;*
7. $P^* = P$*, i.e. P is Hermite'sch;*

Proof. $(4 \Rightarrow 5)$ is trivial.

$(5 \Rightarrow 6)$ Because of $\operatorname{Bild} P \ni Ph \perp h - Ph \in \operatorname{Kern} P$ we have $\|h\|^2 = \|Ph\|^2 + \|h - Ph\|^2$ and thus $\|Ph\| \leq \|h\|$.

$(4 \Leftarrow 6)$ We have $h - Ph = (1-P)h \in \operatorname{Bild}(1-P) = \operatorname{Kern} P$. For $h \perp \operatorname{Kern} P$ we get $0 = \langle h - Ph, h\rangle = \|h\|^2 - \langle Ph, h\rangle$, and hence $\|h\|^2 = \langle Ph, h\rangle \leq \|Ph\|\,\|h\| \leq \|h\|^2$. Furthermore $\|Ph\| = \|h\| = \sqrt{\langle Ph, h\rangle}$ and $\|h-Ph\|^2 = \|h\|^2 - 2\Re(\langle Ph, h\rangle) + \|Ph\|^2 = 0$ for those h. I.e. $(\operatorname{Kern} P)^\perp \subseteq \operatorname{Kern}(1-P) = \operatorname{Bild} P$.
Conversely, let $h \in \operatorname{Bild} P$. Then $h = h_0 + h_1$ with $h_0 \in \operatorname{Kern} P$ and $h_1 \in (\operatorname{Kern} P)^\perp \subseteq \operatorname{Bild} P$. Thus $h_0 = h - h_1 \in \operatorname{Bild} P \cap \operatorname{Kern} P = \{0\}$, i.e. $h_0 = 0$ and $h = h_1 \in (\operatorname{Kern} P)^\perp$. By orthogonalization we get the desired equation.

$(4 \Rightarrow 7)$ Since $v = Pv + (1-P)v$ and $w = Pw + (1-P)w$ with $Pv, Pw \in \operatorname{Bild} P$ and $(1-P)v, (1-P)w \in \operatorname{Kern} P = (\operatorname{Bild} P)^\perp$ we have

$$\begin{aligned}\langle v, P^*w\rangle = \langle Pv, w\rangle &= \langle Pv, Pw + (1-P)w\rangle = \langle Pv, Pw\rangle + 0\\ &= \langle Pv + (1-P)v, Pw\rangle = \langle v, Pw\rangle\end{aligned}$$

hence $P = P^*$.

$(4 \Leftarrow 7)$ since $\operatorname{Kern} P = \operatorname{Kern} P^* = (\operatorname{Bild} P)^\perp$. □

One can even project on some plane orthogonal to w but with rays parallel to some other vector w'.

4.3.2 Perspective Projection

This is a more realistic projection as approximately used by cameras. Again we project to some plane, but this time with rays starting from some 'location' (on the side of the plane opposite to the objects). The simplest situation is, when

the location is at the origin, and the plane we project to is given by $z = 1$. The projection is then given by

$$\langle v_1, v_2, v_3 \rangle \mapsto \langle \frac{v_1}{v_3}, \frac{v_2}{v_3}, 1 \rangle$$

Unfortunately this is not longer a linear mapping (because of the $1/v_3$ term). In order to describe this again by a matrix we consider homogeneous coordinates, i.e. we extend the coordinates of points in $\mathbb{R}^3$ by appending a forth coordinate 1. And on the other hand we associate to each vector in $\mathbb{R}^4$ (with non-vanishing last coordinate) that point in $\mathbb{R}^3$, which we get by dividing the first 3 coordinates by the forth. More generally we identify two vectors $(v, v' \neq 0)$ in $\mathbb{R}^4$ if they are collinear, i.e. there exists a number $\lambda \in \mathbb{R}$ such that $v' = \lambda\, v$. Up to this identification we can write this projection as

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} \frac{v_1}{v_3} \\ \frac{v_2}{v_3} \\ 1 \\ 0 \end{pmatrix} \sim \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{pmatrix}$$

Similar formulas hold for arbitrary projection planes (and location).

The projection onto the plane $z = d$ is given by:

$$\begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

or, equivalently, by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

If we translate 0 and the plane to $\langle 0, 0, -d \rangle$ the projection becomes

$$\begin{aligned} v \mapsto v + d * z \mapsto \left\langle \frac{d\, v_1}{v_3 + d}, \frac{d\, v_2}{v_3 + d}, d \right\rangle \mapsto \left\langle \frac{d\, v_1}{v_3 + d}, \frac{d\, v_2}{v_3 + d}, 0 \right\rangle \\ = \left\langle \frac{v_1}{v_3/d + 1}, \frac{v_2}{v_3/d + 1}, 0 \right\rangle \end{aligned}$$

For $d \to \infty$ this is the orthographic projection from above.

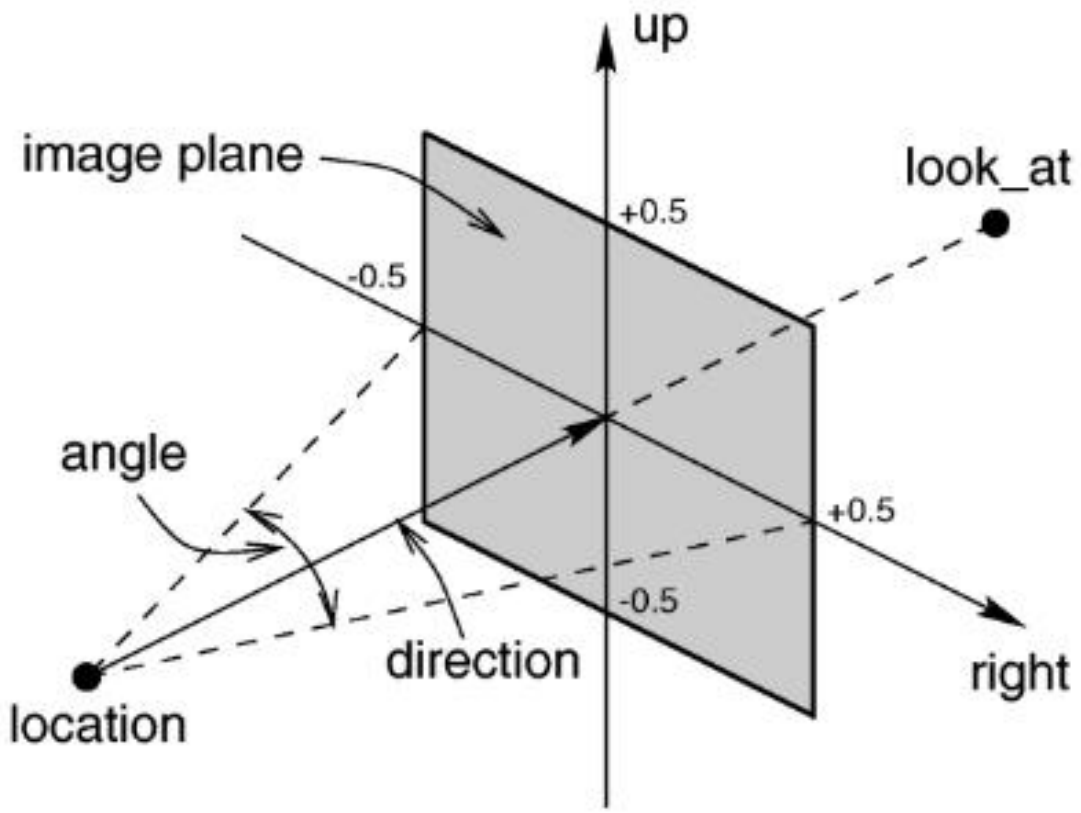
Let the projection plane be given in normal form by $F := \{v : \langle v|n\rangle = d\}$, where n is a unit normal vector to the plane and d is the signed distance to 0, i.e. is positive if n points from 0 to the plane and negative otherwise. The corresponding perspective projection is given by $v \mapsto v'$, where $v' = \lambda\, v \in F$, i.e. $\lambda \langle v|n\rangle = d$, thus

$$v \mapsto v' := \frac{d}{\langle v|n\rangle} v.$$

In homogeneous coordinates this can be described by the matrix

$$\begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ n_1 & n_2 & n_3 & 0 \end{pmatrix}$$

Since we have not infinite large images but some finite rectangle we have to clip the image, and hence we may clip the objects to some prism or pyramid.



Pov-Ray's camera definition

The syntax in **Pov-Ray** for the general camera definition is as follows:

```
CAMERA:
    camera{ [CAMERA_ITEMS...] }
CAMERA_ITEM:
    CAMERA_TYPE | CAMERA_VECTOR | CAMERA_MODIFIER |
    CAMERA_IDENTIFIER
CAMERA_TYPE:
    perspective | orthographic | fisheye | ultra_wide_angle |
    omnimax | panoramic | cylinder CylinderType | spherical
CAMERA_VECTOR:
    location <Location> | right <Right> | up <Up> |
    direction <Direction> | sky <Sky>
CAMERA_MODIFIER:
    angle HORIZONTAL [VERTICAL] | look_at <Look_At> |
    blur_samples Num_of_Samples | aperture Size |
    focal_point <Point> | confidence Blur_Confidence |
    variance Blur_Variance | NORMAL | TRANSFORMATION

Camera default values:

DEFAULT CAMERA:
  camera {
    perspective
    location <0,0,0>
    direction <0,0,1>
    right 1.33*x
    up y
    sky <0,1,0>
  }

CAMERA TYPE: perspective
angle      : ~67.380 ( direction_length=0.5*
                       right_length/tan(angle/2) )
confidence : 0.9 (90%)
direction  : <0,0,1>
focal_point: <0,0,0>
location   : <0,0,0>
look_at    : z
right      : 1.33*x
sky        : <0,1,0>
```

```
up          : y
variance    : 1/128
```

From [pov:180.html#target_713]

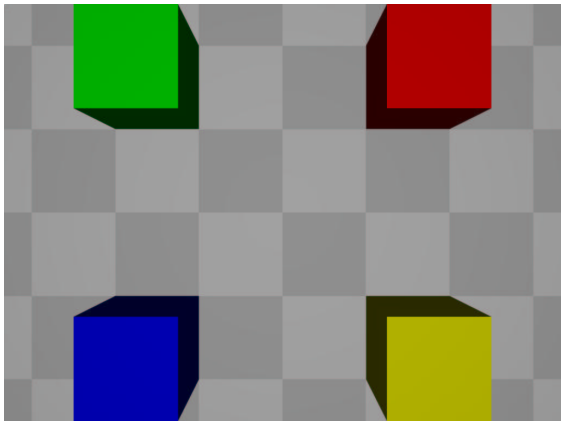
Note also the following:

```
6.4.1  Placing the Camera
  6.4.1.1    Location and Look_At
6.4.1.2    The Sky Vector
6.4.1.3    Angles
6.4.1.4    The Direction Vector
6.4.1.5    Up and Right Vectors
  6.4.1.5.1    Aspect Ratio
6.4.1.5.2    Handedness
6.4.1.6    Transforming the Camera
6.4.2  Types of Projection
6.4.3  Focal Blur
6.4.4  Camera Ray Perturbation
6.4.5  Camera Identifiers
```

4.3.3 Further Projections in Pov-Ray

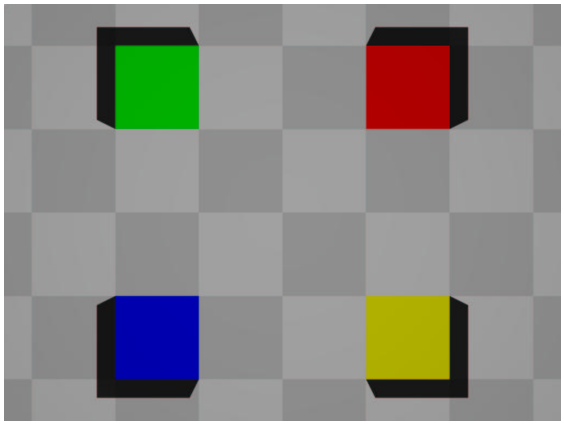
There are several types of projections available in **Pov-Ray**, see the documentation [pov:182.html#target_730] for more details:

- Perspective projection



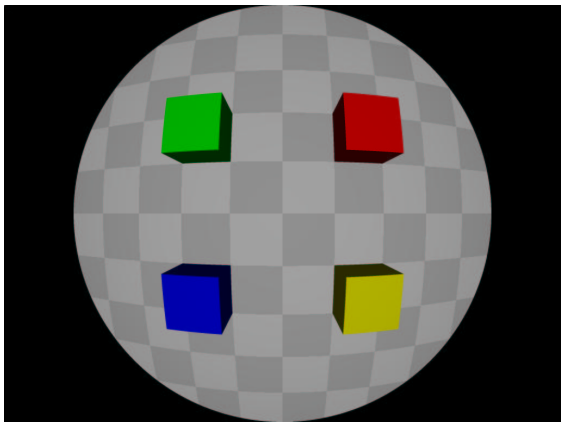
Perspective projection

- Orthographic projection



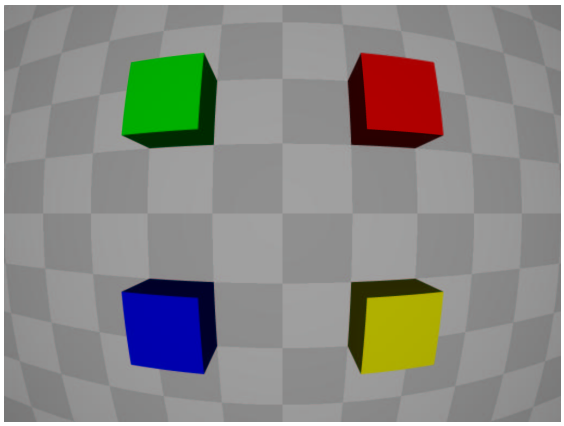
Orthographic projection

- Fisheye projection: A spherical projection. This time we project radially onto a sphere and then by some projection the sphere to the plan. Uses Polar-coordinates.



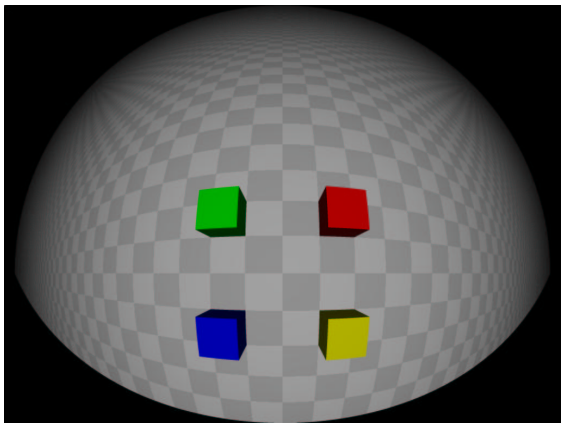
Fisheye projection

- Ultra wide angle projection: Here we project onto a cylinder and then we flatten the cylinder



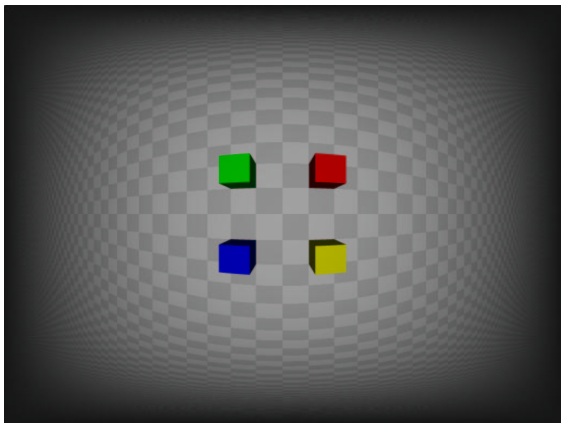
Ultrawideangle projection

- Omnimax projection: Is similar to a 180^o-fisheye projection but with aspect ratio unequal to 1.



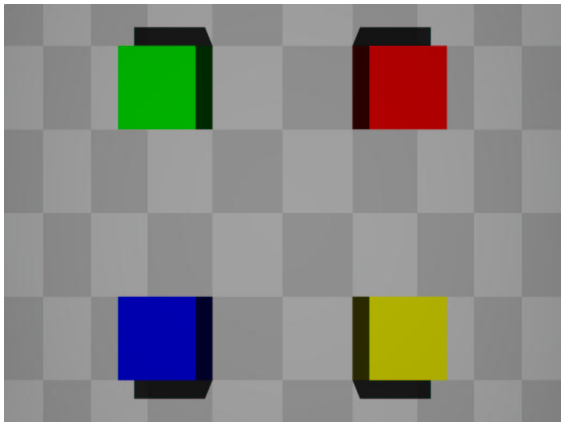
Omnimax projection

- Panoramic projection: Cylindrical equirectangular projection



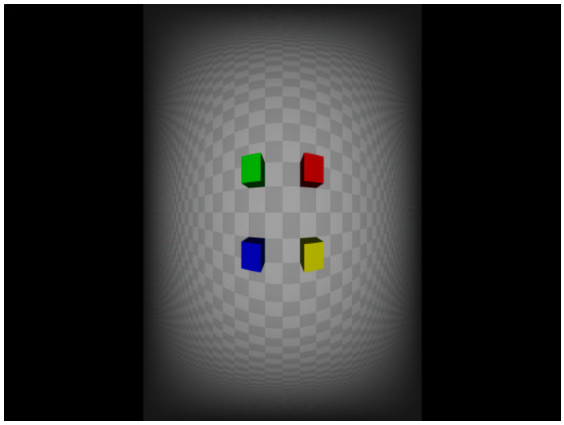
Panoramic projection

- Cylindrical projection: Project onto a vertical or horizontal cylinder with respect to a center or the cylinders axes.



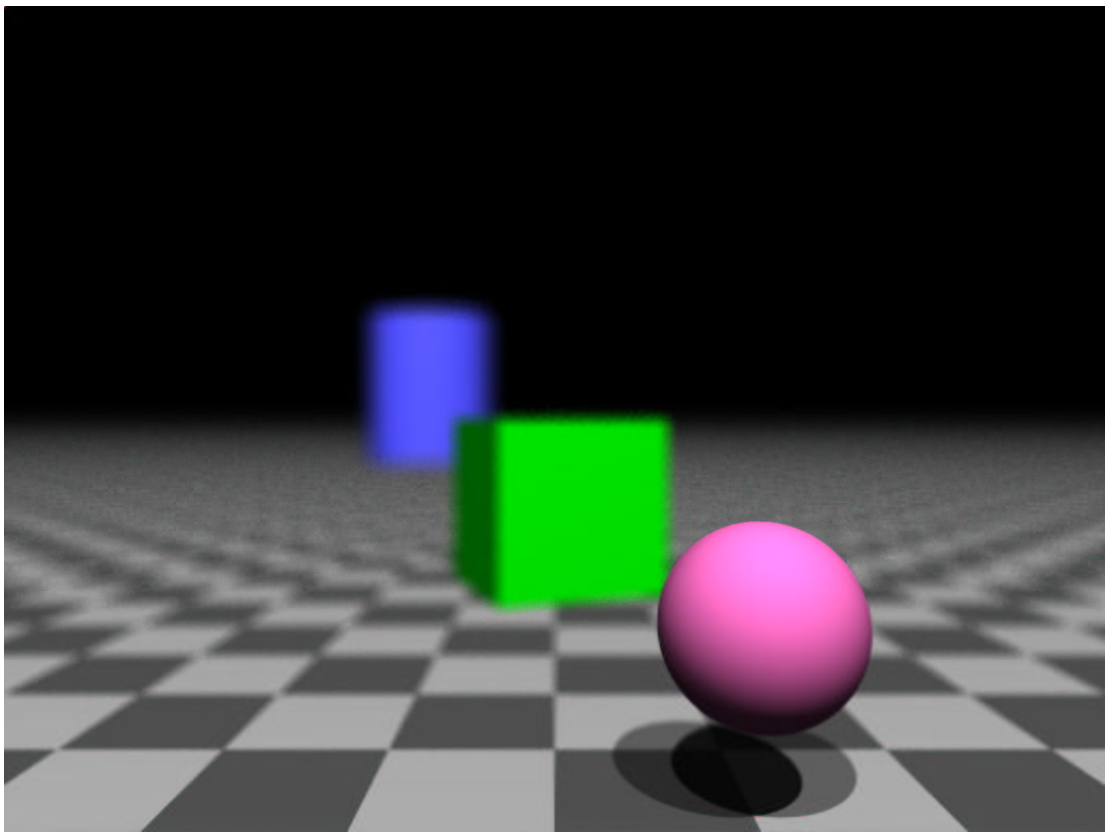
Cylindrical projection

- Spherical projection: Project onto a sphere and uses rectangular coordinates.



Spherical projection

Using the optional parameters `focal_point`, `aperture` and `blur_samples` one can change the by default infinite field of depth to some realistic range.



Finite field of depth

```
camera {
  location <0.0, 1.0, -10.0>
  look_at  <0.0, 1.0,  0.0>
  focal_point < 1, 1, -6>    // pink sphere in focus
  aperture 0.4     // a nice compromise
  blur_samples 20      // more samples, higher quality image
}
```

Chapter 5

Objects

In this chapter we are going to describe the objects we want to depict.

Links

- Objects:
- dtrg: POV-Ray Manual: 5. Scene language description reference
  5.2.0-3.html
- Florian Fischer: Primitives in Pov-Ray 3 (Einfache Objekte)
  objectpv.htm
- objects/blob
- :POV-Ray: Documentation: 3.6.1 Blob Object
  POV-Ray: Documentation: 3.6.1 Blob Object.html
- Rune Johansen: blob tutorial
  blobs.asp.html
- Jeff Lee: Understanding Blobs in POV-Ray
  blobs.html

  Objects/isosurface:
- St. Benge: Geometric Creations - Isosurface Tutorial - Beginning
  iso_tutorial1.html

- St. Benge: Geometric Creations - Isosurface Tutorial - Part Two
  iso_tutorial2.html
- St. Benge: Geomectric Creations - Isosurface Tutorial - Part 3
  iso_tutorial3.html
- Smellenbergh: Isosurface manual: Overview
  isotutrs/iso_ind.html

Splines:

- Paul Heckbert: addendum.pdf
  almond.srv.cs.cmu.edu/. . . /addendum.pdf
- Kenneth I. Joy: Online geometric modeling notes
  graphics.cs.ucdavis.edu/. . . /index.html
- Kenneth I. Joy: Bezier-Curves.pdf
  graphics.cs.ucdavis.edu/. . . /Bezier-Curves.pdf
  keywords: Historics on Bezier curves

★ Kenneth I. Joy: Arbitrary-Bezier-Curves.pdf
graphics.cs.ucdavis.edu/. . . /Arbitrary-Bezier-Curves.pdf
keywords: short description of Bezierer splines (using Bernstein polynomials)

- Kenneth I. Joy: Bezier-Control-Polygons-for-a-Cubic-Curve.pdf
  graphics.cs.ucdavis.edu/. . . /Bezier-Control-Polygons-for-a-Cubic-Curve.pdf
  keywords: pieceing together Bezier curves
- Kenneth I. Joy: B-Spline-Curve-Definition.pdf
  graphics.cs.ucdavis.edu/. . . /B-Spline-Curve-Definition.pdf
- Kenneth I. Joy: Cubic-Refinement-Eigenvalues.pdf
  graphics.cs.ucdavis.edu/. . . /Cubic-Refinement-Eigenvalues.pdf
- Kenneth I. Joy: Cubic-Subdivision-Curve-Direct-mathcalculation.pdf
  graphics.cs.ucdavis.edu/. . . /Cubic-Subdivision-Curve-Direct-mathcalculation.pdf
- Kenneth I. Joy: Cubic-Uniform-B-Spline-Curve-Splitting.pdf
  graphics.cs.ucdavis.edu/. . . /Cubic-Uniform-B-Spline-Curve-Splitting.pdf
- Kenneth I. Joy: Deboor-Cox-mathcalculation.pdf
  graphics.cs.ucdavis.edu/. . . /Deboor-Cox-mathcalculation.pdf

- Kenneth I. Joy: Divide-and-Conquer-Bezier-Curve.pdf
  graphics.cs.ucdavis.edu/. . . /Divide-and-Conquer-Bezier-Curve.pdf
- Kenneth I. Joy: Catmull-Clark.pdf
  graphics.cs.ucdavis.edu/. . . /Catmull-Clark.pdf
- Kenneth I. Joy: Chaikins-Algorithm.pdf
  graphics.cs.ucdavis.edu/. . . /Chaikins-Algorithm.pdf
- Kenneth I. Joy: Control-Points.pdf
  graphics.cs.ucdavis.edu/. . . /Control-Points.pdf
- Kenneth I. Joy: Cubic-Bezier-Curves.pdf
  graphics.cs.ucdavis.edu/. . . /Cubic-Bezier-Curves.pdf
- Kenneth I. Joy: Cubic-B-Spline-Curve-Refinement.pdf
  graphics.cs.ucdavis.edu/. . . /Cubic-B-Spline-Curve-Refinement.pdf
- Kenneth I. Joy: The-Support-of-a-Blending-Function.pdf
  graphics.cs.ucdavis.edu/. . . /The-Support-of-a-Blending-Function.pdf
- Kenneth I. Joy: Uniform-B-Splines-as-a-Convolution.pdf
  graphics.cs.ucdavis.edu/. . . /Uniform-B-Splines-as-a-Convolution.pdf
- Kenneth I. Joy: Uniform-Normalized-Blending-Functions.pdf
  graphics.cs.ucdavis.edu/. . . /Uniform-Normalized-Blending-Functions.pdf
- Kenneth I. Joy: Uniform-Two-Scale-Proof.pdf
  graphics.cs.ucdavis.edu/. . . /Uniform-Two-Scale-Proof.pdf
- Kenneth I. Joy: Uniform-Two-Scale-Relation.pdf
  graphics.cs.ucdavis.edu/. . . /Uniform-Two-Scale-Relation.pdf
- Kenneth I. Joy: Vertex-and-Edge-Points.pdf
  graphics.cs.ucdavis.edu/. . . /Vertex-and-Edge-Points.pdf

  Surfaces:

- Kenneth I. Joy: Bezier-Patches.pdf
  graphics.cs.ucdavis.edu/. . . /Bezier-Patches.pdf
- Kenneth I. Joy: Arbitrary-Bezier-Patch.pdf
  graphics.cs.ucdavis.edu/. . . /Arbitrary-Bezier-Patch.pdf
  keywords: Bezier patch for surface

- Kenneth I. Joy: Bezier-Curves-on-Bezier-Patches.pdf
  graphics.cs.ucdavis.edu/. . . /Bezier-Curves-on-Bezier-Patches.pdf
  keywords: Bézier patches as families of Bézier curves
- Kenneth I. Joy: Bezier-Patch-Subdivision.pdf
  graphics.cs.ucdavis.edu/. . . /Bezier-Patch-Subdivision.pdf
- Kenneth I. Joy: Cubic-B-Spline-Surface-Refinement.pdf
  graphics.cs.ucdavis.edu/. . . /Cubic-B-Spline-Surface-Refinement.pdf
- Kenneth I. Joy: Loop-Surfaces.pdf
  graphics.cs.ucdavis.edu/. . . /Loop-Surfaces.pdf
- Kenneth I. Joy: Matrix-Cubic-Bezier-Curve.pdf
  graphics.cs.ucdavis.edu/. . . /Matrix-Cubic-Bezier-Curve.pdf
- Kenneth I. Joy: Matrix-Cubic-Bezier-Patch.pdf
  graphics.cs.ucdavis.edu/. . . /Matrix-Cubic-Bezier-Patch.pdf
- Kenneth I. Joy: Quadratic-Bezier-Curves.pdf
  graphics.cs.ucdavis.edu/. . . /Quadratic-Bezier-Curves.pdf
- Kenneth I. Joy: Quadratic-B-Spline-Curve-Refinement.pdf
  graphics.cs.ucdavis.edu/. . . /Quadratic-B-Spline-Curve-Refinement.pdf
- Kenneth I. Joy: Quadratic-B-Spline-Surface-Refinement.pdf
  graphics.cs.ucdavis.edu/. . . /Quadratic-B-Spline-Surface-Refinement.pdf
- Kenneth I. Joy: Refinement.pdf
  graphics.cs.ucdavis.edu/. . . /Refinement.pdf
- Kenneth I. Joy: Reparameterizing-Bezier-Curves.pdf
  graphics.cs.ucdavis.edu/. . . /Reparameterizing-Bezier-Curves.pdf
- Kenneth I. Joy: Subdivision-Curves.pdf
  graphics.cs.ucdavis.edu/. . . /Subdivision-Curves.pdf
- Kenneth I. Joy: Subdivision-Surfaces.pdf
  graphics.cs.ucdavis.edu/. . . /Subdivision-Surfaces.pdf

Tutorials:

- dtrg:
  www-sfb288.math.tu-berlin.de/. . . /index.html

- Ken Joy: Geometric Modeling – On-Line Notes
  graphics.cs.ucdavis.edu/. . . /index.html
- Friedrich A. Lohmueller: POV-Ray descriptions, tutorials and samples for the POV-Ray raytracing program for beginners and advanced users
  www.f-lohmueller.de/. . . /pov_eng.htm
- Friedrich A. Lohmueller: POV-Ray Tutorial - Einführung, deutsche Beschreibungen, Anleitungen und Beispiele zum Raytracer POVRAY zu 3D-Grafik mittels Raytracing
  www.f-lohmueller.de/. . . /pov_ger.htm
- Robet B. Chaffe: POV-Ray SDL Quick Reference
  quickref.html
  keywords: povray quick reference
- Michiel van de Panne: computer graphics lecture topics
  www.cs.ucla.edu/. . . /lectureTopics.html

  Geometry/Splines:
- Frank Pfenning: 10-splines.pdf
  almond.srv.cs.cmu.edu/. . . /10-splines.pdf
- Frank Pfenning: 10-splines.pdf (color)
  almond.srv.cs.cmu.edu/. . . /10-splines.pdf

  Geometry/Surfaces:
- Heckbert-15462: curves_implicit_2.pdf
  almond.srv.cs.cmu.edu/. . . /curves_implicit_2.pdf

  Geometry/Parametrizations:
- Frank Pfenning: 09-curves.pdf
  almond.srv.cs.cmu.edu/. . . /09-curves.pdf
- Frank Pfenning: 09-curves.pdf (color)
  almond.srv.cs.cmu.edu/. . . /09-curves.pdf
- Heckbert-15462: curves_param_2.pdf
  almond.srv.cs.cmu.edu/. . . /curves_param_2.pdf
- Heckbert-15462: geomod.pdf
  almond.srv.cs.cmu.edu/. . . /geomod.pdf

- www.f-lohmueller.de/. . . /povshpe1e.htm
- www.f-lohmueller.de/. . . /povshpe2e.htm

Pov-Ray-Documentation:

- [pov:3.2] Basic Shapes
- [pov:3.3] CSG Objects
- [pov:3.4] Spline Based Shapes
- [pov:3.5] Polygon Based Shapes
- [pov:3.6] Other Shapes
- [pov:6.5] Objects
- [pov:10.1.8] Objects

We will lead our way through the huge zoo of objects starting from the most special ones and reach the most general mathematic descriptions at the end. The objects available in **Pov-Ray** are of different types: finite solid objects, finite patch objects, infinite objects, iso- and parametric surfaces, and objects obtained by CSG (constructive solid geometry). See [pov:395.html#target_1783] for a short overview of the syntax.

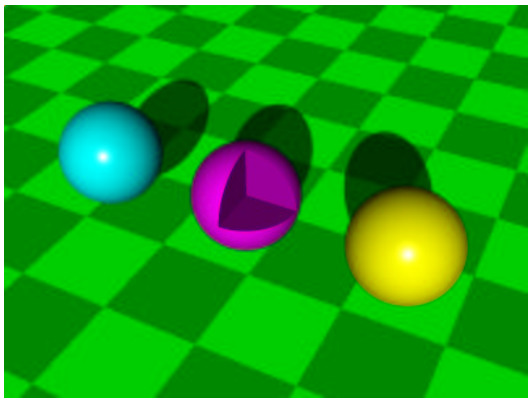
5.1 Solid Finite Objects

In this group are the finite (i.e. bounded) solid objects.

5.1.1 Sphere

The syntax of the SPHERE object is:

```
SPHERE:
  sphere { CENTER, RADIUS [SPHERE_MODIFIERS] }

SPHERE_MODIFIERS:
  [UV_MAPPING] & [OBJECT_MODIFIERS]
```



Spheres

See also:

- [pov:6.5.1.9] Sphere
- [pov:3.1.4] Sphere Object
- www.f-lohmueller.de/. . . /sphere1e.htm

The object described is a sphere (or more correctly a ball, and not just the surface) with the 3d-vector CENTER(= $\langle c_1, c_2, c_3 \rangle$) as center and with the scalar RADIUS(= r) as radius:

$$\{\langle x_1, x_2, x_3 \rangle : (x_1 - c_1)^2 + (x_2 - c_2)^2 + (x_3 - c_3)^2 \leq r^2\}.$$

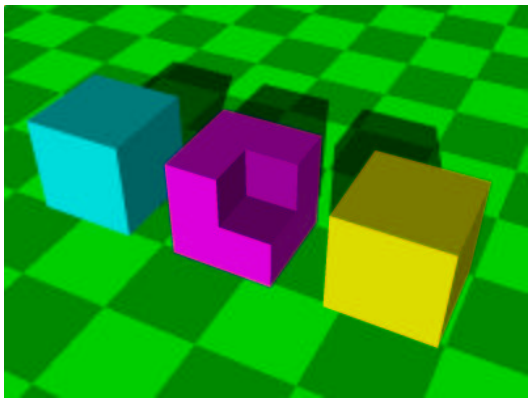
Each object may contain transformations as part of the OBJECT_MODIFIERS, so a general sphere may be described by the default sphere with CENTER=0 and RADIUS=1 as follows:

```
sphere { c, r } = sphere { 0, 1 scale r translate c }
```

5.1.2 Box

The syntax of the BOX object is:

```
BOX:
  box { CORNER1, CORNER2 [BOX_MODIFIERS] }

BOX_MODIFIERS:
 [UV_MAPPING] & [OBJECT_MODIFIERS]
```



Boxes

See also:

- [pov:6.5.1.2] Box
- [pov:3.2.1] Box Object
- www.f-lohmueller.de/. . . /box1e.htm
- blobs.asp.html

The object described is a cube with sides (of possibly different length) parallel to the axes and two opposite vertices CORNER1(=c) and CORNER2(=d):

$$\{\langle x_1, x_2, x_3\rangle : x_1 \in \overline{c_1 d_1},\ x_2 \in \overline{c_2 d_2},\ x_3 \in \overline{c_3 d_3}\},$$

where $\overline{c_i d_i}$ is the segment $\{(1-t)\,c_i + t\,d_i : 0 \le t \le 1\}$ from c_i to d_i.

Again

```
box { c1, c2 } = box { 0, 1 scale c2-c1 translate c1 }
```

In order to depict a box with sides being not parallel to the axes we have to apply some rotation to a box as above:

```
box { c1, c2 rotate v }
```

5.1.3 Superellipsoid

A shape between sphere and box is the SUPERELLIPSOID. The syntax of the superquadric ellipsoid object is:

```
SUPERELLIPSOID:
  superellipsoid { < K1, K2 > [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.1.11] Superquadric Ellipsoid
- [pov:3.6.5] Superquadric Ellipsoid Object

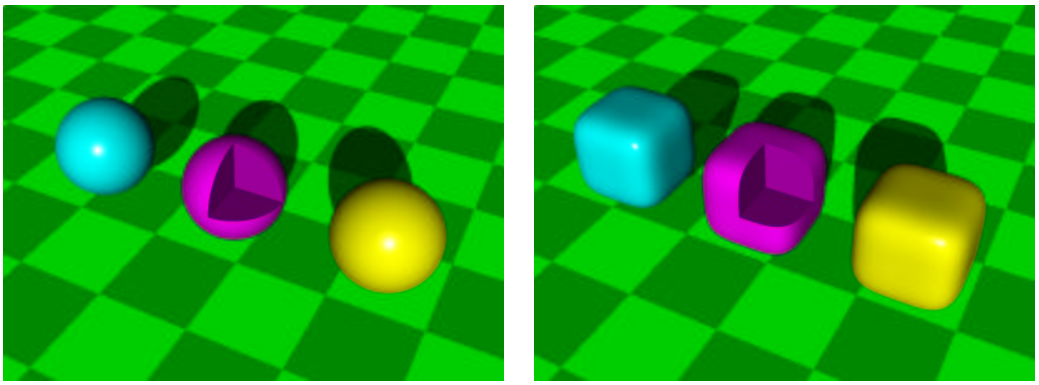
This object is given via two parameters K1= $e > 0$ and K2= $n > 0$ by

$$\{\langle x,y,z\rangle : \left\|(\|(x,y)\|_{2/e}, z)\right\|_{2/n}^{2/n} = \left(|x|^{2/e} + |y|^{2/e}\right)^{e/n} + |z|^{2/n} == 1\},$$

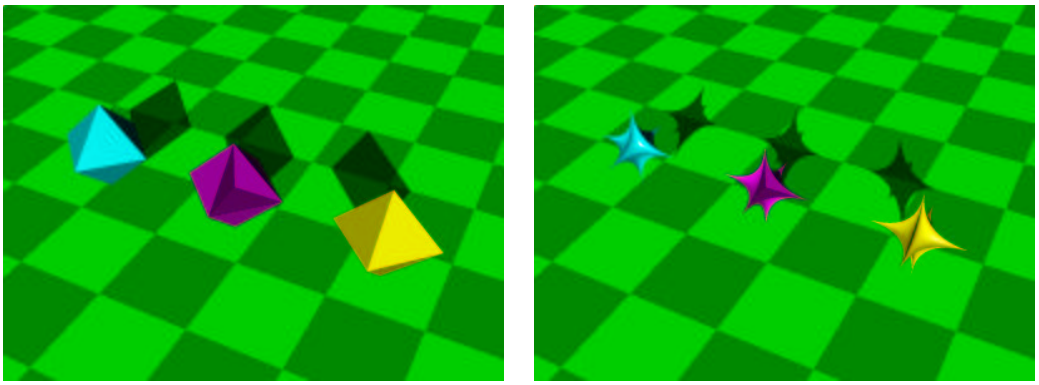
where the p-norm of a 2d-vector $\langle x,y\rangle$ is given by

$$\|\langle x,y\rangle\|_p := \begin{cases} (|x|^p + |y|^p)^{1/p} & \text{for } 1 \le p < \infty \\ \max\{|x|,|y|\} & \text{for } p = \infty \end{cases}$$

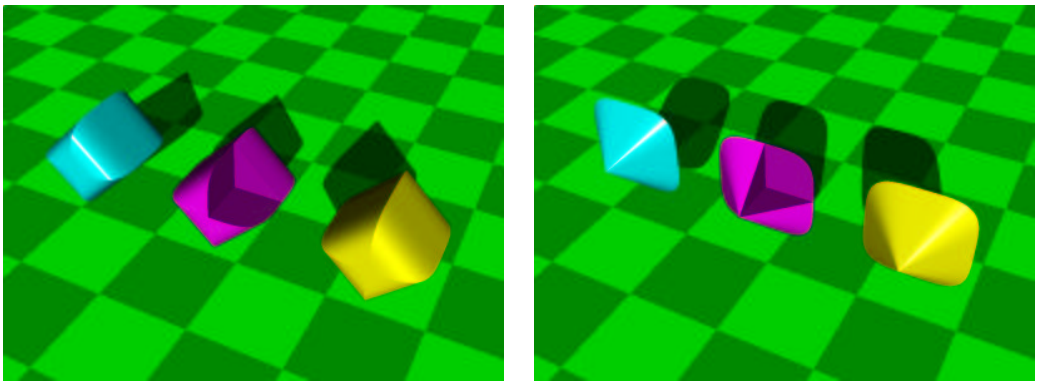
Hence we get a cube for $e = n = 0$ and a sphere for $e = n = 1$.



Superellipsoids with $e = n = 1$ and with $e = n = 1/2$



Superellipsoids with $e = n = 2$ and with $e = n = 4$



Superellipsoids with $e = 2, n = 1/2$ and with $e = 1/2, n = 2$

5.1.4 Cylinder

The syntax of the CYLINDER object is:

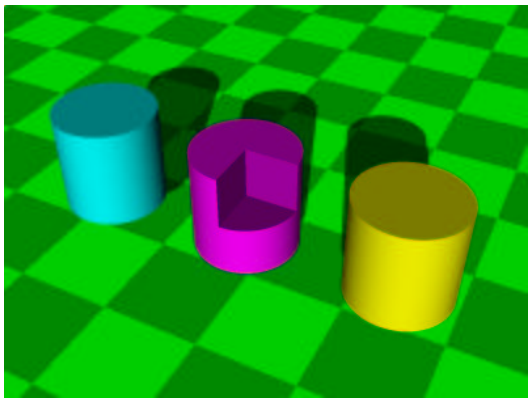
```
CYLINDER:
  cylinder { BASE_CENTER, CAP_CENTER, RADIUS [open]
    [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.1.4] Cylinder
- [pov:3.2.3] Cylinder Object
- www.f-lohmueller.de/. . . /cylindr1e.htm

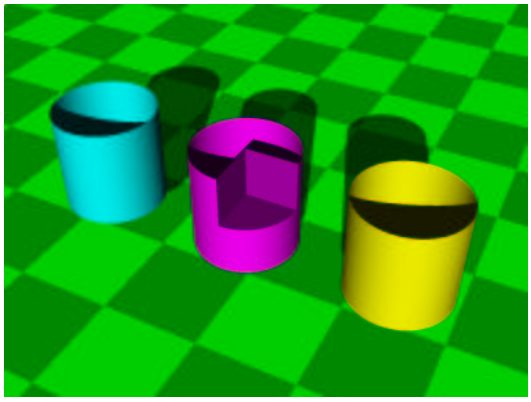
This describes a circular cylinder orthogonal to the axes from BASE_CENTER (=V0) to the CAP_CENTER (=V1) with radius RADIUS (=R), e.g.

$$\texttt{cylinder}\{0, y, R\} = \{\langle X, Y, Z\rangle : X^2 + Z^2 \leq R^2,\ 0 \leq Y \leq 1\}$$



Cylinders

The optional keyword `open` makes the cylinder hollow and removes the base and cap disk.



Open Cylinders

The general cylinder can be obtained from 'cylinder { 0, y, 1 }' by scaling, rotation and translation.

5.1.5 Cone

The syntax of the CONE object is:

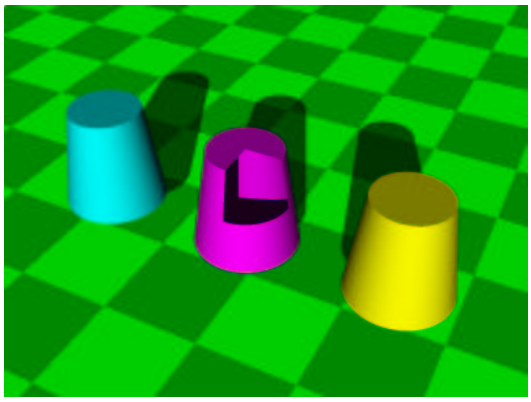
```
CONE:
  cone { BASE_CENTER, BASE_RADIUS, CAP_CENTER, CAP_RADIUS [open]
    [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.1.3] Cone
- [pov:3.2.2] Cone Object
- www.f-lohmueller.de/. . . /cone1e.htm

This describes a circular cone orthogonal to the axes from BASE_CENTER (=V0) with radius BASE_RADIUS (=R0) to the CAP_CENTER (=V1) with radius CAP_RADIUS (=R1), e.g.

$$\texttt{cone}\{0, R0, y, R1\} = \{\langle X, Y, Z\rangle : X^2 + Z^2 \leq (R0 + Z\,(R1 - R0))^2,\ 0 \leq Y \leq 1\}$$



Cones

A cylinder is the special case of the cone where the two radii are equal.

5.1.6 Prism

The syntax of the PRISM object is:

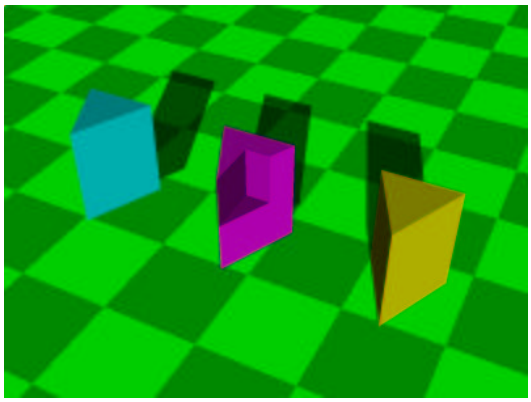
```
PRISM:
  prism { [PRISM_SPLINE_TYPE] [PRISM_SWEEP_TYPE]
HEIGHT0, HEIGHT1, NUM_POINTS, POINT_LIST [open] [PRISM_MODIFIERS] }

PRISM_SPLINE_TYPE:
  linear_spline | quadratic_spline | cubic_spline | bezier_spline

PRISM_SWEEP_TYPE:
  linear_sweep | conic_sweep
```

```
PRISM_MODIFIERS:
  [sturm [BOOL]] | [OBJECT_MODIFIERS]
```

See also:

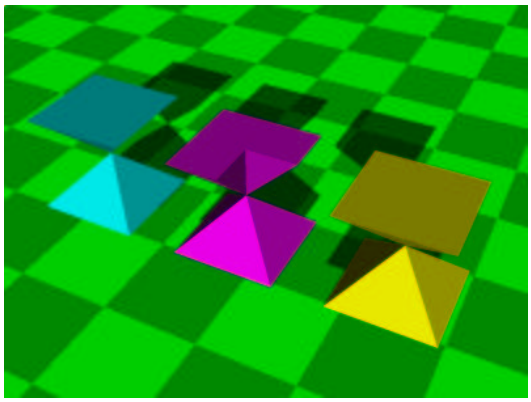
- [pov:6.5.1.8] Prism
- [pov:3.4.3] Prism Object
- www.f-lohmueller.de/. . . /prism1e.htm

This describes a pyramid whose base is given by NUM_POINTS many 2d-points listed in POINT_LIST and it extends in y-direction from height HEIGHT0 to HEIGHT1.



Prismas

The type PRISM_SPLINE_TYPE determines how the points are connected by splines (see (5.1.7)) and the keyword `conic_sweep` instead of `linear_sweep` for the PRISM_SWEEP_TYPE makes a pyramid with base at $Y = 1$ and vertex at 0.



Pyramids

5.1.7 Interpolation and Splines

We need curves passing through a finite number of given points. In algebra we learn that there is exactly one polynomial p of degree at most n passing through $n+1$ points

$$P_0 = \langle t_0, x_0 \rangle, \ldots, P_n = \langle t_n, x_n \rangle$$

with pairwise different $t_0, t_1, \ldots, t_n$. A simple formula for obtaining this polynomial p is the LAGRANGE INTERPOLATION FORMULA:

$$p(x) = \sum_{k=0}^{n} x_k \, L_k(t) \text{ where } L_k(t) := \prod_{j \neq k} \frac{t - t_j}{t_k - t_j}$$

This is easily checked, since $L_k(t_j) = 0$ for $k \neq j$ and $L_k(t_k) = 1$.

For a large number of points this becomes quite lengthy to calculate and for t between the given t_j the values can be quite far away from the x_j. So the idea is to interpolate a fixed small number of successive points and piece them together.

The simplest way would be to use linear interpolation of successive points and we would thus obtain a polygon.

But we could also take 3 (, 4 or more) successive points and take the quadratic (, cubic, ...) polynomial connecting these and piece them together.

The disadvantage of this method will be that at the points, where we paste the pieces together the curve may take sharp turns, i.e. the left-sided and right-sided derivatives may be different.

To avoid this problem Bezier curves have been invented. In order to discuss them we need the BERNSTEIN POLYNOMIALS:

$$B_k^n(t) := \binom{n}{k} t^k (1-t)^{n-k},$$

e.g.

$$\begin{gathered} B_0^0(t) = 1; \\ B_0^1(t) = (1-t), \qquad B_1^1(t) = t; \\ B_0^2(t) = (1-t)^2, \qquad B_1^2(t) = 2t(1-t), \qquad B_2^2(t) = t^2; \\ B_0^3(t) = (1-t)^3, \qquad B_1^3(t) = 3t(1-t)^2, \qquad B_2^3(t) = 3t^2(1-t), \qquad B_3^3(t) = t^3 \\ \vdots \end{gathered}$$

A recursive definition is

$$B_k^n(t) := (1-t)B_k^{n-1} + tB_{k-1}^{n-1}(t).$$

We have:

- $B_k^n(t) \geq 0$ for all $0 \leq t \leq 1$.
- $\sum_{k=0}^n B_k^n(t) = 1$.
- $B_k^n(t) = \sum_{i=k}^n (-1)^{i-k} \binom{n}{i} \binom{i}{k} t^i$.
- $t^k = \sum_{i=k-1}^{n-1} \binom{i}{k} / \binom{n}{k} B_i^n(t)$.
- From the last two equations we deduce that $(B_k^n)_{0 \leq k \leq n}$ is a basis of the vector space of all polynomials of degree $\leq n$.
- $(B_k^n)'(t) = n(B_{k-1}^{n-1}(t) - B_k^{n-1}(t))$
- A formula for a linear combination $B(t) = \sum_{k=0}^n c_j B_j^n(t)$ of the Bernstein polynomials of degree 2 is given by

$$B(t) = (1, t, t^2) \cdot \begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_2 \\ c_2 \end{pmatrix}.$$

Now the BEZIER CURVE of degree n given by $n+1$ many points $P_0, \ldots, P_n$ is given by

$$P(t) = \sum_{j=0}^{n} P_j \, B_i^n(t)$$

This can be obtained recursively by

$$\begin{aligned} P_i^0(t) &:= P_i \\ P_i^j(t) &:= (1-t) P_{i-1}^{j-1}(t) + t P_i^{j-1}(t) \\ P(t) &:= P_n^n(t) \end{aligned}$$

A geometric interpretation of this can be found in
graphics.cs.ucdavis.edu/.../Subdivision-Curves.pdf

The main properties of the Bezier curve P are the following:

- P is polynomial (of degree at most n) and hence C^∞.
- $P(0) = P_0$ and $P(1) = P_n$.
- The tangent to P at 0 is the line $\overline{P_0 P_1}$.
- The tangent to P at 1 is the line $\overline{P_{n-1} P_n}$.

- The curve P lies in convex hull of $\{P_0, P_1, \ldots, P_n\}$, note, however, that the points $P_1, \ldots, P_{n-1}$ will not ly on P. They are only points which control the shape of the curve P.

5.1.8 Lathe

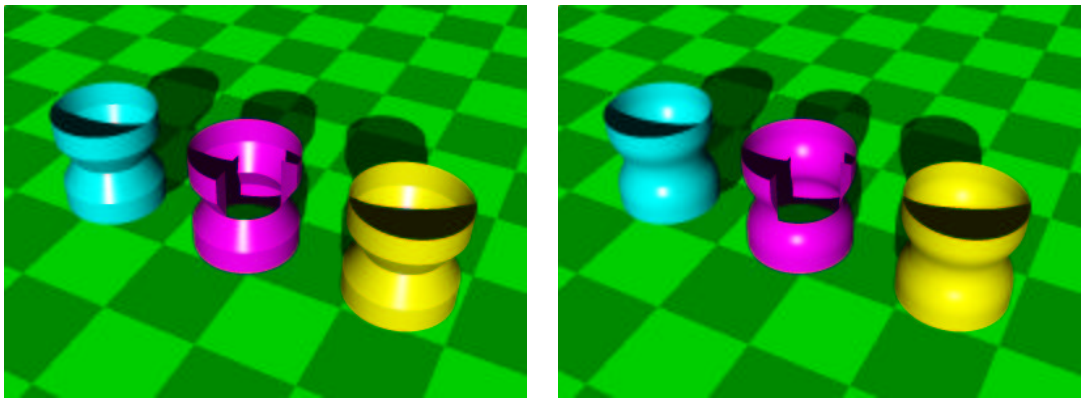
The syntax of a LATHE object is:

```
LATHE:
  lathe { [LATHE_SPLINE_TYPE] NUM_POINTS, POINT_LIST
    [LATHE_MODIFIERS] }

LATHE_SPLINE_TYPE:
  linear_spline | quadratic_spline | cubic_spline | bezier_spline

LATHE_MODIFIERS:
  [sturm [BOOL]] | [UV_MAPPING] | [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.1.7] Lathe
- [pov:3.4.1] Lathe Object; See also for intro to splines

This describes an object obtained by rotating the area between the y-axes and the spline given by the NUM_POINTS many 2d-points in POINT_LIST around the y-axes.



Lathe objects with linear and with Bezier splines

For the spline type `linear_spline` we need NUM_POINTS≥ 2, and for the spline type `quadratic_spline` we need n =NUM_POINTS≥ 3, where from P_k to P_{k+1} we use the quadratic spline constructed for P_{k-1}, P_k, P_{k+1}. Thus the curve will start only at P_1 and end at P_n, and P_0 is just a control point.

For `cubic_splines` we need n =NUM_POINTS≥ 4, where from P_k to P_{k+1} we use the cubic spline constructed for $P_{k-1}, P_k, P_{k+1}, P_{k+2}$. Thus the curve will start only at P_1 and end at P_{n-1}, and both P_0 and P_n are just control points.

Finally for `bezier_spline` we need $4n$ =NUM_POINTS, where we use the cubic Bezier curve through the points $P_{4n}, P_{4n+1}, P_{4n+2}, P_{4n+3}$ from P_{4n} to $P_{4n+3} = P_{4(n+1)}$. Thus only the points $P_0, P_3 = P_4, \ldots, P_{4n-1}$ will ly on the curve, the others are just control points.

5.1.9 Sor

Another version of a surface of revolution is the SOR object. Its syntax is:

```
SOR:
  sor { NUM_POINTS, POINT_LIST [open] [SOR_MODIFIERS] }

SOR_MODIFIERS:
  [sturm [BOOL]] & [UV_MAPPING] & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.1.12] Surface of Revolution
- [pov:3.4.2] Surface of Revolution Object
- www.f-lohmueller.de/. . . /sor1e.htm

This object is obtained by rotating the area between y-axes and the cubic polynomial from P_k to P_{k+1} through the points P_{k-1}, P_k, P_{k+1} and P_{k+2} around the y-axes. The point P_0 and P_n are thus only control points.

See also:

- [pov:3.4.2] Surface of Revolution Object

Advantage of `sor` over `lathe` is that the intersection test for `sor` needs to solve a cubic polynomial whereas that for `lathe` needs to solve a polynomial of degree 6, which means quite a lot more work.

5.1.10 Torus

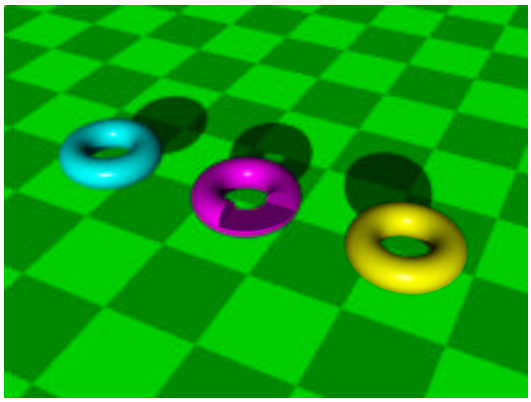
The syntax of the TORUS object is:

```
TORUS:
  torus { MAJOR_RADIUS, MINOR_RADIUS [TORUS_MODIFIERS] }

TORUS_MODIFIERS:
  [sturm [BOOL]] & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.1.14] Torus
- [pov:3.2.5] Torus Object
- www.f-lohmueller.de/. . . /torus1e.htm

This is the particular case of a surface of revolution, where a disk of radius MINOR_RADIUS($= r$) and center at the x-axes in distance MAJOR_RADIUS($= R$) from 0 is rotated around the y-axes.



Tori

In parametric form this surface is given by latitude $\theta \in [-\pi, \pi]$ and longitude $\varphi \in [0, 2\pi]$ as

$$\langle \theta, \varphi \rangle \mapsto \langle (R + r\cos\theta)\cos\varphi, (R + r\cos\theta)\sin\varphi, r\sin\theta \rangle$$

or implicitly by

$$(\sqrt{x^2 + z^2} - R)^2 + y^2 = r^2.$$

5.1.11 Blob

The mathematical objects discussed so far are not well suited for modeling natural forms like a human hand for example. Although the digits are roughly speaking cylinder connecting the joints which are approximately spheres, the transition between the spheres and the cylinders should be smooth. This can be modeled by the BLOB object. The syntax of a blob object is:

```
BLOB:
  blob { [threshold FLOAT] BLOB_ITEM ... BLOB_ITEM
    [BLOB_MODIFIERS] }

BLOB_ITEM:
  sphere { CENTER, RADIUS, [strength] STRENGTH
    [COMPONENT_MODIFIERS] } |
  cylinder { BASE_CENTER, CAP_CENTER, RADIUS,
    STRENGTH [COMPONENT_MODIFIERS] }

COMPONENT_MODIFIERS:
  [TEXTURE] & [PIGMENT] & [NORMAL] & [FINISH] & [TRANSFORMATION...]

BLOB_MODIFIERS:
  [hierarchy [BOOL]] & [sturm [BOOL]] & [OBJECT_MODIFIERS]
```

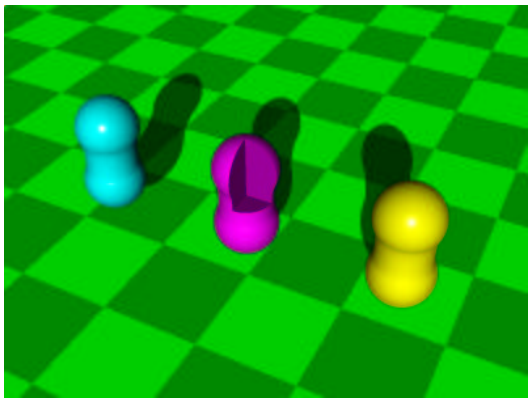
See also:

- [pov:6.5.1.1] Blob
- [pov:3.6.1] Blob Object
- Rune Johansen: runevision :: rune's blob tutorialrune—vision :: rune's blob tutorial
  blobs.asp.html
- Jeff Lee: Understanding Blobs in POV-Ray
  blobs.html

A point in space belongs to this object if the sum of its densities

$$\text{density} = \text{strength} * \left(1 - \left(\frac{\text{distance}}{\text{radius}}\right)^2\right)^2$$

with respect to all the BLOB_ITEM's is at least the thresh hold THRESH_HOLD.



Blobs

5.1.12 Sphere-Sweep

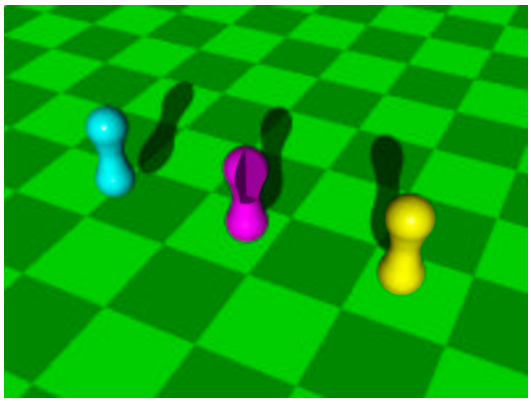
Another object that can be used to model smooth transitions between different spheres is the SPHERE SWEEP object. Its syntax is:

```
SPHERE_SWEEP:
  sphere_sweep { SWEEP_SPLINE_TYPE NUM_SPHERES, SPHERE_ITEM, ... ,
    SPHERE_ITEM [tolerance F_DEPTH_TOLERANCE] [OBJECT_MODIFIERS] }

SWEEP_SPLINE_TYPE:
  linear_spline | b_spline | cubic_spline

SPHERE_ITEM:
  CENTER, RADIUS
```

See also:

- [pov:6.5.1.10] Spheresweep
- [pov:3.4.4] Sphere Sweep Object

This describes a union of spheres whose centers are formed by the spline through the centers of the SPHERE_ITEM's and the radii are given by the values given by the spline through the radii of the SPHERE_ITEM's.



Sphere-sweep objects

5.1.13 Height-Field

The syntax of a HEIGHT FIELD object is:

```
HEIGHT_FIELD:
  height_field { HF_IMAGE [HF_MODIFIERS] }

HF_IMAGE:
  FUNCTION_IMAGE | [HF_TYPE] FILE_NAME

HF_TYPE:
  gif | tga | pot | png | pgm | ppm | jpeg | tiff | sys

HF_MODIFIERS:
  [hierarchy [BOOL]] & [smooth [BOOL]] & [water_level FLOAT]
    & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.1.5] Height Field
- [pov:3.6.2] Height Field Object
- www.f-lohmueller.de/. . . /povheigh1e.htm

This is given by the set of points $\langle X, Y, Z \rangle$ with $0 \leq X \leq 1$ and $0 \leq Z \leq 1$, where Y is at most the value of the (interpolate) pixel in the given image of dimension WIDTH $*$ HEIGHT at the scaled point in column $X *$ WIDTH and row $Y *$ HEIGHT.

5.1.14 Julia-Fractal

There is also the JULIA FRACTAL object, whose syntax is:

```
JULIA_FRACTAL:
  julia_fractal { 4D_VECTOR [JF_ITEMS] [OBJECT_MODIFIERS] }

JF_ITEMS:
  [ALGEBRA_ITEM] & [max_iteration INT] & [precision FLOAT]
    & [slice V4_NORMAL, F_DISTANCE]

ALGEBRA_ITEM:
  quaternion [QUATER_FUNCTION] | hypercomplex [HYPER_FUNCTION]

QUATER_FUNCTION:
  sqr | cube

HYPER_FUNCTION:
  sqr | cube | exp | reciprocal | sin | asin | sinh | asinh
    | cos | acos | cosh | acosh | tan | atan | tanh | atanh
    | ln | pwr (FLOAT,FLOAT)
```

See [pov:6.5.1.6] Julia Fractal if you need this.

5.1.15 Text

Another more useful object is the TEXT object, with syntax:

```
TEXT:
  text { ttf FILE_NAME STRING THICKNESS, OFFSET
    [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.1.13] Text
- [pov:3.4.6] Text Object

This allows to insert a line STRING of text as 3-dimensional letters in the font given by FILE_NAME and with depth THICKNESS.

5.2 Finite Patch Objects

These are objects described by the following 2-dimensional (=totally thin) fine shapes:

```
FINITE_PATCH_OBJECT:
  BICUBIC_PATCH | DISC | MESH | MESH2 | POLYGON
    | TRIANGLE | SMOOTH_TRIANGLE
```

See also:

- [pov:6.5.2] Finite Patch Primitives

5.2.1 Triangle

The most basic one among these is the TRIANGLE object, with syntax:

```
TRIANGLE:
  triangle { CORNER1, CORNER2, CORNER3 [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.2.6] Triangle and Smooth Triangle

This describes a triangle with the three 3d-vectors CORNER1 ($= C_1$), CORNER2 ($= C_2$), and CORNER3 ($= C_3$) as vertices, i.e.

$$\{\sum_{i=1}^{3} t_i\, C_i : t_1, T_2, T_3 \geq 0, \sum_{i=1}^{3} t_i = 1\}.$$

5.2.2 Smooth Triangle

There exists a variant, the SMOOTH TRIANGLE object, were the virtual surface normal (see (6.3)) at the vertices are also given. Its syntax is:

```
SMOOTH_TRIANGLE:
  smooth_triangle {
  CORNER1, NORMAL1, CORNER2, NORMAL2, CORNER3, NORMAL3
[OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.2.6] Triangle and Smooth Triangle

5.2.3 Polygon

More general is the POLYGON object, with syntax:

```
POLYGON:
  polygon { NUM_POINTS, POINT, ...,  POINT [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.2.5] Polygon
- [pov:3.5.3] Polygon Object

One has to make sure, that the POINTS all ly in one plane. This is easiest by taking on coordinate constant (say 0) and rotate the polygon afterwards to the required position.

5.2.4 Mesh

To build more complex objects formed by lots of triangles one can use the MESH object, with syntax:

```
MESH:
  mesh { MESH_TRIANGLE ... MESH_TRIANGLE [MESH_MODIFIERS] }

MESH_TRIANGLE:
```

```
  triangle { CORNER1, CORNER2, CORNER3 [MESH_UV_VECTORS]
    [MESH_TEXTURE] } |
  smooth_triangle {
CORNER1, NORMAL1, CORNER2, NORMAL2, VCORNER3, VNORMAL3
[MESH_UV_VECTORS] [MESH_TEXTURE] }

MESH_UV_VECTORS:
  uv_vectors PARAM1, PARAM2, PARAM3

MESH_TEXTURE:
  texture { TEXTURE_IDENTIFIER } |
  texture_list { TEXTURE_IDENTIFIER ... TEXTURE_IDENTIFIER }

MESH_MODIFIERS:
  [inside_vector V_DIRECTION] & [hierarchy [BOOL]] & [UV_MAPPING]
    & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.2.3] Mesh
- [pov:3.5.1] Mesh Object

5.2.5 Mesh2

A more efficient way of listing a list of triangles, by specifying all vertices and only the indices of the vertices of all triangles, is given by the MESH2 object, with syntax:

```
MESH2:
  mesh2 { MESH2_VECTORS [TEXTURE_LIST] MESH2_INDICES
    [MESH2_MODIFIERS] }

MESH2_VECTORS:
  VERTEX_VECTORS [NORMAL_VECTORS] [UV_VECTORS]

VERTEX_VECTORS:
  vertex_vectors { NUM_VERTICES, VECTOR [, VECTOR]... }

NORMAL_VECTORS:
```

```
  normal_vectors { NUM_NORMALS, VECTOR [, VECTOR]... }

UVECTORS:
  uv_vectors { NUM_UVECTORS, 2D_VECTOR [, 2D_VECTOR]... }

TEXTURE_LIST:
  texture_list { NUM_TEXTURES, TEXTURE [, TEXTURE]... }

MESH2_INDICES:
  FACE_INDICES [NORMAL_INDICES] [UINDICES]

FACE_INDICES:
  face_indices { NUM_FACES, FACE_INDICES_ITEM
    [, FACE_INDICES_ITEM]... }
FACE_INDICES_ITEM:
  VECTOR [, TEXTURE_INDEX [, TEXTURE_INDEX, TEXTURE_INDEX ]]
NORMAL_INDICES:
  normal_indices { NUM_FACES, VECTOR [, VECTOR]... }
UINDICES:
  uv_indices { NUM_FACES, VECTOR [, VECTOR]... }
MESH2_MODIFIERS:
  [inside_vector DIRECTION] & [UMAPPING] & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.2.4] Mesh2
- [pov:3.5.2] Mesh2 Object

5.2.6 Bicubic Patch

A 2-dimensional variant of cubic splines is the BICUBIC PATCH object with syntax:

```
BICUBIC_PATCH:
  bicubic_patch { PATCH_ITEMS [PATCH_UV_VECTORS] CONTROL_POINTS
    [BICUBIC_PATCH_MODIFIERS] }

PATCH_ITEMS:
  type PATCH_TYPE & [u_steps INT] & [v_steps INT]
    & [flatness FLOAT]
```

```
PATCH_TYPE:
  0 | 1

PATCH_UV_VECTORS:
  uv_vectors V2_CORNER1, V2_CORNER2, V2_CORNER3, V2_CORNER4

CONTROL_POINTS:
  16 VECTORs, optionally separated by commas.

BICUBIC_PATCH_MODIFIERS:
  [UV_MAPPING] & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.2.1] Bicubic Patch
- [pov:3.4.5] Bicubic Patch Object

5.2.7 Disk

The syntax of the DISK object is:

```
DISC:
  disc { CENTER, NORMAL, RADIUS[, HOLE_RADIUS] [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.2.2] Disc

which describes a disc with center CENTER normal to the 3d-vector NORMAL and with radius RADIUS (and a hole with radius HOLE_RADIUS)

5.3 Infinite Shapes

These objects are potentially infinite(=unbounded) solid shapes.

```
INFINITE_SOLID_OBJECT:
  PLANE | POLY | CUBIC | QUARTIC | QUADRIC
```

See also:

- [pov:6.5.3] Infinite Solid Primitives

5.3.1 Plane

The syntax of the PLANE object is:

```
PLANE:
  plane { NORMAL, DISTANCE [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.3.1] Plane
- [pov:3.2.4] Plane Object
- www.f-lohmueller.de/. . . /plane1e.htm

This describes the plane (or better half-space) with surface normal NORMAL= $\langle a_1, a_2, a_3 \rangle$ and (signed) distance DISTANCE(= d) from 0 given by

$$\{\langle X, Y, Z \rangle : a_1 \cdot X + a_2 \cdot Y + a_3 \cdot Z \leq d\}.$$

5.3.2 Quadric

The syntax of the QUADRIC object is:

```
QUADRIC:
  quadric { COEFF1, COEFF2, COEFF3, COEFF4 [OBJECT_MODIFIERS] }
```

See also:

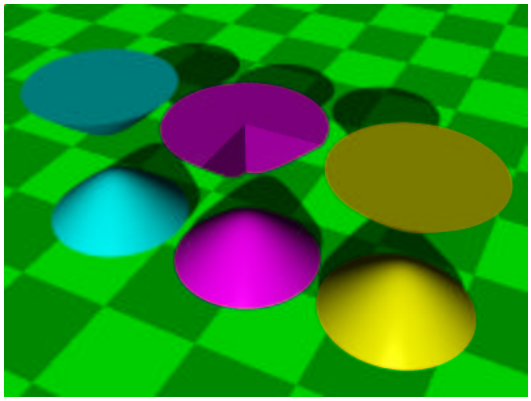
- [pov:6.5.3.3] Quadric

It describes a quadric (surface given by a quadratic polynomial in the variables X, Y and Z) with coefficients COEFF1(= $\langle a_X, a_Y, a_Z \rangle$) of the homogeneous quadratic part, coefficients COEFF2(= $\langle b_Z, b_Y, b_X \rangle$) of the inhomogeneous quadratic part,

coefficients COEFF3($= \langle c_X, c_Y, c_Z \rangle$) of the homogeneous linear part and constant coefficient COEFF4($= d$) given by

$$\{\langle X, Y, Z \rangle : a_X X^2 + a_Y Y^2 + a_Z Z^2 + b_Z XY + b_Y XZ + b_X YZ + \\ + c_X X + c_Y Y + c_Z Z + d \leq 0\}$$

Note the unusual order of COEFF2! As particular cases we get ellipsoids, paraboloids, hyperboloids, and the degenerated cases of cylinders, double cones, etc. ...



Hyperboloids

5.3.3 Cubic

The corresponding object of degree 3 is the CUBIC object, with syntax

```
CUBIC:
  cubic { < CUBIC_COEFFICIENTS > [POLY_MODIFIERS] }

CUBIC_COEFFICIENTS:
  20 FLOATs separated by commas.
```

See also:

- [pov:6.5.3.2] Poly, Cubic and Quartic

For the order of the coefficients see POLY below.

5.3.4 Quartic

The corresponding object of degree 4 is the QUARTIC object, with syntax

```
QUARTIC:
  quartic { < QUARTIC_COEFFICIENTS > [POLY_MODIFIERS] }

QUARTIC_COEFFICIENTS:
  35 FLOATs separated by commas.
```

See also:

- [pov:6.5.3.2] Poly, Cubic and Quartic

For the order of the coefficients see POLY below. An example of such a quartic is the equation for the torus (obtained by isolating in the formula for the torus above the term $2R\sqrt{X^2+Z^2}$ on one side and taking the square of both sides):

$$4(X^2+Z^2)R^2 = (X^2+Z^2+Y^2-r^2)^2.$$

5.3.5 Poly

The general object of this type of degree 2 $\leq$ORDER$\leq$ 15 is the POLY object with syntax:

```
POLY:
  poly { ORDER, < POLY_COEFFICIENTS > [POLY_MODIFIERS] }

POLY_COEFFICIENTS:
  A quantity n of FLOATs separated by commas,
    where n is ((ORDER+1)*(ORDER+2)*(ORDER+3))/6.

POLY_MODIFIERS:
  [sturm [BOOL]] & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.3.2] Poly, Cubic and Quartic
- [pov:3.6.4] Poly Object

It describes the object given by

$$\left\{\langle X,Y,Z\rangle : \sum_{i=0}^{d} X^{d-i} \sum_{j=0}^{i} Y^{i-j} \sum_{k=0}^{j} Z^{j-k}\, a_{i,j,k} \le 0\right\}$$

where the coefficients are given in exactly this order. Note that this is not the usual ordering by degree of homogeneity. But the inverse lexicographical ordering of the sequence of exponents of X, Y and Z. Thus for degree 3 the 20 monomials are ordered as

$$\begin{array}{cccccccc}
X^3Y^0Z^0 & & & & & & & \\
X^2Y^1Z^0 & X^2Y^0Z^1 & X^2Y^0Z^0 & & & & & \\
X^1Y^2Z^0 & X^1Y^1Z^1 & X^1Y^1Z^0 & X^1Y^0Z^2 & X^1Y^0Z^1 & X^1Y^0Z^0 & & \\
X^0Y^3Z^0 & X^0Y^2Z^1 & X^0Y^2Z^0 & X^0Y^1Z^2 & X^0Y^1Z^1 & X^0Y^1Z^0 & & \\
X^0Y^0Z^3 & X^0Y^0Z^2 & X^0Y^0Z^1 & X^0Y^0Z^0 & & & &
\end{array}$$

The number of coefficients of a polynomial of degree at most ORDER($= d$) in the three variables X, Y and Z has as many coefficients, as there are triples (i,j,k) with $i+j+k \le d$ with $i,j,k \ge 0$. Such triple can be equally described by three numbers $0 \le k \le j \le i \le d$ as in the sum above. Such 3 numbers can be viewed as 3 separators at 3 different positions among $d+3$ many items, where k is the number of non-separators to the left of the first one, j is that of non-separators to the left of the second and i is that of non-separators to the left of the third one. By combinatorics the number of such choices is

$$\binom{n+3}{3} = \frac{(n+3)(n+2)(n+1)}{6}.$$

5.4 Isosurface and Parametric Surface

5.4.1 Isosurface

A general method of describing an object is by an equation as implicitly given surface, called ISOSURFACE in **Pov-Ray**. Its syntax is

```
ISOSURFACE:
  isosurface { FLOAT_USER_FUNCTION [ISOSURFACE_ITEMS]
    [OBJECT_MODIFIERS] }

ISOSURFACE_ITEMS:
  [contained_by { CONTAINER }] & [threshold FLOAT]
    & [accuracy FLOAT] & [max_gradient FLOAT
    [evaluate MIN_ESTIMATE, MAX_ESTIMATE,
    ATTENUATION]] & [open] & [INTERSECTION_LIMIT]

CONTAINER:
  sphere { CENTER, RADIUS } |
box { CORNER1, CORNER2 }

INTERSECTION_LIMIT:
  max_trace INT | all_intersections
```

See also:

- [pov:6.5.4] Isosurface Object
- [pov:3.6.3] Isosurface Object
- objects/isosurface
- St. Benge: Geometric Creations - Isosurface Tutorial - Part 3 iso_tutorial3.html :
- St. Benge: Geometric Creations - Isosurface Tutorial - Beginning iso_tutorial1.html :
- St. Benge: Geometric Creations - Isosurface Tutorial - Part Two iso_tutorial2.html :
- :http://3dgallery.dhs.org/tutorials/iso_tutor.html :
- Smellenbergh: Isosurface manual: Overview isotutrs/iso_ind.html :
- :POV-Ray: Documentation: 6.5.4 Isosurface Object :

It is given as

$$\{\langle X, Y, Z\rangle : \text{FLOAT\_USER\_FUNCTION}(X, Y, Z) = \text{THRESHOLD} \\ \text{for all } (X, Y, Z) \in \text{CONTAINER}.\}$$

The FLOAT_USER_FUNCTION is a user defined function f which returns some float value for any point $\langle X, Y, Z\rangle$ in 3d-space. Note that mathematically we can guarantee (by the implicit function theorem) that the corresponding surface f^{-1}(THRESHOLD) is non-singular only in points P, where f is C^1 and the derivative $f'(P)$ is non-vanishing.

With the optional CONTAINER one can restrict the object to a box or sphere. The other optional parameters control the numeric process used in solving the implicit equation.

5.4.2 Parametric

Another method to describe a surface is by a parametrization. The syntax of the corresponding PARAMETRIC object is

```
PARAMETRIC:
  parametric { USER_FUNCTION_X, USER_FUNCTION_Y, USER_FUNCTION_Z
    CORNER1, CORNER2 [PARAMETRIC_ITEMS] [OBJECT_MODIFIERS] }

PARAMETRIC_ITEMS:
  [contained_by { CONTAINER } ] & [max_gradient FLOAT]
    & [accuracy FLOAT] & [precompute I_DEPTH, x, y, z]

CONTAINER:
  sphere { V_CENTER, F_RADIUS } | box { V_CORNER1, V_CORNER2 }
```

See also:

- [pov:6.5.5] Parametric Object

This describes the object given by the image of the function

$$(u, v) \mapsto \langle f_X(u, v), f_Y(u, v), f_Z(u, v)\rangle,$$

where f_X, f_Y , f_Z are the USER_FUNCTION's and (u, v) varies in the 2-dimensional box given by two opposite corners CORNER1 and CORNER2.

5.5 Constructive Solid Geometry

We can describe complex shapes by set-theoretic operations from already described objects. The objects obtained that way are one of:

```
CSG_OBJECT:
  UNION | INTERSECTION | DIFFERENCE | MERGE
```

See also:

- [pov:6.5.6] Constructive Solid Geometry
- [pov:3.3] CSG Objects
- [pov:3.3.1] What is CSG?
- www.f-lohmueller.de/. . . /povcsg1e.htm

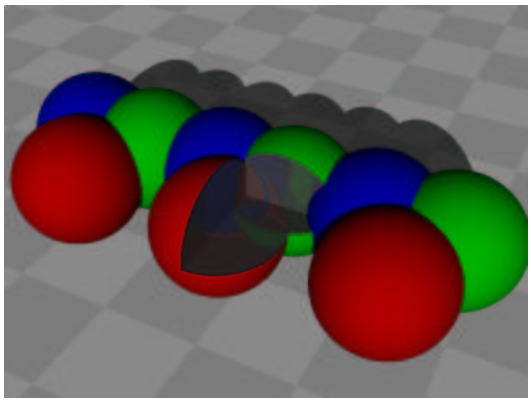
5.5.1 Union

The syntax of a UNION is:

```
UNION:
  union { UNION_OBJECT ... UNION_OBJECT [UNION_MODIFIERS] }

UNION_OBJECT:
  OBJECT | LIGHT

UNION_MODIFIERS:
  [split_union BOOL] & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.6.2] Union
- [pov:3.3.2] CSG Union



Unions of spheres

5.5.2 Merge

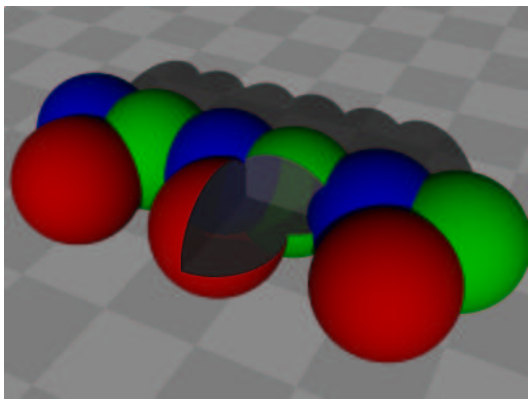
There is a variant of union namely MERGE, with syntax:

```
MERGE:
  merge { SOLID_OBJECT SOLID_OBJECT... [OBJECT_MODIFIERS] }
```

See also:

- [pov:6.5.6.5] Merge
- [pov:3.3.5] CSG Merge
- www.f-lohmueller.de/. . . /povcsg2e.htm

The difference between `merge` and `union` is that in a `merge` the internal parts of surfaces are removed.



Merged spheres

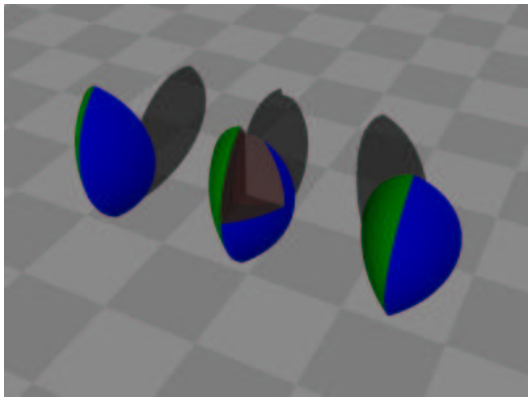
5.5.3 Intersection

The syntax of an INTERSECTION is:

```
INTERSECTION:
  intersection { SOLID_OBJECT ... SOLID_OBJECT
    [INTERSECTION_MODIFIERS] }

SOLID_OBJECT:
  FINITE_SOLID_OBJECT | INFINITE_SOLID_OBJECT | ISOSURFACE
    | CSG_OBJECT

INTERSECTION_MODIFIERS:
  [cutaway_textures] & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.6.3] Intersection
- [pov:3.3.3] CSG Intersection



Intersections of spheres

5.5.4 Difference

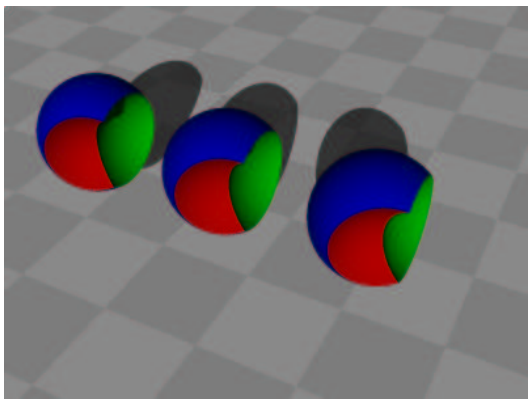
The syntax of a DIFFERENCE is:

```
DIFFERENCE:
  difference { SOLID_OBJECT ... SOLID_OBJECT
    [DIFFERENCE_MODIFIERS] }
```

```
DIFFERENCE_MODIFIERS:
  [cutaway_textures] & [OBJECT_MODIFIERS]
```

See also:

- [pov:6.5.6.4] Difference
- [pov:3.3.4] CSG Difference



Differences of spheres

5.6 Lights

- Light cf. [FvDFH90, 13]
- Rendering Techniques cf. [FvDFH90, 14]
- Illumination and Shading cf. [FvDFH90, 16]
  Ambient light, diffuse Reflection, Light-source attenuation, Atmospheric Attenuation, Specular Reflection, texture maps, bump maps, shadows, transparency, refraction
- Hugo Elias: A Physical Model Of Light
  hugo.elias/. . . /x_physic.htm
  Reflection, refraction, color, light sources:
- Hugo Elias: Radiosity
  hugo.elias/. . . /radiosity.htm
  Radiosity, direct and global illumination:

- Hugo Elias: Motion Blur
  hugo.elias/. . . /x_motion.htm
- Hugo Elias: The Phong Approximation
  hugo.elias/. . . /x_polyph.htm
  Phong shading:
- Hugo Elias: Fast Phong Shading
  hugo.elias/. . . /x_polyp2.htm
- Hugo Elias: Bump Mapping
  hugo.elias/. . . /x_polybm.htms
- Hugo Elias: Simple Shadow Casting
  hugo.elias/. . . /x_shadow.htm
- Frank Pfenning: 08-shading.pdf
  almond.srv.cs.cmu.edu/. . . /08-shading.pdf
  almond.srv.cs.cmu.edu/. . . /08-shading.pdf
  Gouraud/Phong-shading, light sources, normals in OpenGL:
- Frank Pfenning: 16-ray.pdf
  almond.srv.cs.cmu.edu/. . . /16-ray.pdf
  almond.srv.cs.cmu.edu/. . . /16-ray.pdf
  forward/backward raytracing, shadow, reflection, ray-intersections, transmitted light, translucency:
- Heckbert-15462: raycasting_2.pdf
  almond.srv.cs.cmu.edu/. . . /raycasting_2.pdf
  forward/backward raytracing, shadow, ray-intersections, radiosity:
- Heckbert-15462: raytracing.pdf
  almond.srv.cs.cmu.edu/. . . /raytracing_2.pdf
  almond.srv.cs.cmu.edu/. . . /raytracing.pdf
  raytracing, refraction, aliasing, supersampling, motion-blur, soft shadows, depth of field:
- Heckbert-15462: visibility_2.pdf
  almond.srv.cs.cmu.edu/. . . /visibility_2.pdf
  Visibility-methods, Gouraud-, Phong-, facet-shading:
- Frank Pfenning: 17-spatial.pdf
  almond.srv.cs.cmu.edu/. . . /17-spatial.pdf
  almond.srv.cs.cmu.edu/. . . /17-spatial.pdf

almond.srv.cs.cmu.edu/.../spatial_2.pdf
bounding volumes, grids, octrees, trees:

- Frank Pfenning: 18-radiosity.pdf
  almond.srv.cs.cmu.edu/.../18-radiosity.pdf
  almond.srv.cs.cmu.edu/.../18-radiosity.pdf
  radiosity:

- Frank Pfenning: 19-global.pdf
  almond.srv.cs.cmu.edu/.../19-global.pdf
  almond.srv.cs.cmu.edu/.../19-global.pdf
  radiosity:

- Heckbert-15462: shading_2.pdf
  almond.srv.cs.cmu.edu/.../shading_2.pdf :

- Heckbert-15462: rendering.pdf
  almond.srv.cs.cmu.edu/.../rendering.pdf
  almond.srv.cs.cmu.edu/.../rendering.pdf
  scan conversion:

- Frank Pfenning: 07-lighting.pdf
  almond.srv.cs.cmu.edu/.../07-lighting.pdf
  almond.srv.cs.cmu.edu/.../07-lighting.pdf
  light sources, ambient light, phong illumination, (diffuse) reflection:

- Paul Heckbert: rad.pdf
  almond.srv.cs.cmu.edu/.../rad.pdf
  almond.srv.cs.cmu.edu/.../rad.pdf
  global illumination and radiosity:

- [pov:3.7] The Light Source
  - [pov:3.7.1] The Pointlight Source
  - [pov:3.7.2] The Spotlight Source
  - [pov:3.7.3] The Cylindrical Light Source
  - [pov:3.7.4] The Area Light Source
  - [pov:3.7.5] The Ambient Light Source
  - [pov:3.7.6] Light Source Specials
  - [pov:6.5.7] Light Sources
  - [pov:6.5.8] Light Groups
  - [pov:10.1.7] Lights

In order to be able to see the objects discussed in the previous chapter we need some light. This is achieved in **Pov-Ray** by the LIGHT_SOURCE statement with syntax:

```
LIGHT_SOURCE:
  light_source { V_LOCATION, COLOR [LIGHT_SOURCE_ITEMS] }

LIGHT_SOURCE_ITEMS:
  [LIGHT_TYPE] & [AREA_LIGHT_ITEMS] & [LIGHT_MODIFIERS]

LIGHT_TYPE:
  spotlight [SPOTLIGHT_ITEMS] |
  cylinder  [SPOTLIGHT_ITEMS]

SPOTLIGHT_ITEMS:
  [radius FLOAT] & [falloff FLOAT] &
  [tightness FLOAT] & [point_at VECTOR]

AREA_LIGHT_ITEMS:
  area_light AXIS1, AXIS2, SIZE1, SIZE2 [AREA_LIGHT_MODIFIERS]

AREA_LIGHT_MODIFIERS:
  [adaptive INT] & [jitter] & [circular] & [orient]

LIGHT_MODIFIERS:
  [LIGHT_PHOTONS] &
  [looks_like { OBJECT }] &
  [TRANSFORMATION...] &
  [fade_distance FLOAT] & [fade_power FLOAT] &
  [media_attenuation [BOOL]] &
  [media_interaction [BOOL]] &
  [shadowless] &
  [projected_through { OBJECT_IDENTIFIER }] &
  [parallel [point_at VECTOR]]

LIGHT_PHOTONS:
  photons { LIGHT_PHOTON_ITEMS }

LIGHT_PHOTON_ITEMS:
  [refraction BOOL] & [reflection BOOL] & [area_light]
```

```
Light source default values:

  LIGHT_TYPE        : pointlight
  falloff           : 70
  media_interaction : on
  media_attenuation : off
  point_at          : <0,0,0>
  radius            : 70
  tightness         : 10
```

5.6.1 Point Lights

This is the most basic light source. It sends light of a given color C from a point S in space uniformly in all directions:

```
  light_source { S, rgb C }
```

The color can be specified by the keyword `rgb` and a 3d-vector $C = \langle R, G, B \rangle$ with red($= R$), green($= G$) and blue($= B$) component.

5.6.2 Spotlights

If we add the keyword `SPOTLIGHT`, then the light is restricted to a cone with tip at S and the line from S to P as axes. The half opening angle of the cone is given by the float R1 and the full prescribed intensity of the light is attained in the cone with the float R0 as half opening angle.

```
light_source { S, rgb C
  spotlight point_at P radius R0 falloff R1 [ tightness T ] }
```

By setting the optional parameter `tightness T` to a value greater than 0 one can soften the light distribution exponentially.

5.6.3 Cylindrical Lights

If we add instead of `spotlight` the keyword `CYLINDER`, then the light is restricted to a cylinder. Note however, that the lightrays are still emitted from the point P and are thus not parallel.

```
light_source { S, rgb C
  cylinder point_at P radius R0 falloff R1 [ tightness T ] }
```

5.6.4 Parallel Lights

We may add the keyword `PARALLEL` to any light source together with `point_at P` to make the light rays parallel to the ray from S to P.

```
light_source { S, rgb C ... parallel point_at P }
```

5.6.5 Area Lights

The light sources discussed so far have the unrealistic feature that they cast hard shadows. Since in reality the light sources are not singular points but have some size (like the sun or a light bulb) the cast a penumbra. To model this we would have to integrate the light over that part of the light source which can be seen from a given point in space. This would be far to slow, thus the following approximate method of AREA LIGHTs can be used.

```
light_source { S, rgb C
  area_light V1, V2, N1, N2 [jitter] [circular] [orient] }
```

This positions a $N1 \times N2$-grid of point lights along the rectangle with corner S and directional vectors $V1$ and $V2$ of its sides.

By using the optional keyword `jitter` these pointlights are randomly displaced a little to make the shadows even softer during calculation of each single pixel.

By using the optional keyword `circular` the rectangle is deformed to an ellipse.

Finally, the optional keyword `orient` takes care that the rectangle is oriented normal to each light ray it emits. This option should only be used together with circular and quadratic area lights.

5.6.6 Light Fading

In reality light diminishes with distance, since the number of photons passing through spheres of varying radii centered at the light source is constant, but the surface of the spheres is proportional to the radius squared. So light intensity is inverse proportional to the square of the distance from the source.

By default povray does not take this into account, but by adding the keywords `FADE_DISTANCE D0` and `FADE_POWER E` one can modify this behavior. The formula for the light intensity reaching an object with distance D from the source with intensity I is given by

$$I \cdot \frac{2}{1 + (D/D0)^E}$$

Note that $E = 2$ corresponds best to the real situation, but here the term '1+' has been inserted in order to avoid ∞ for $D = 0$. The parameter $D0$ is the distance, where this formula gives the prescribed intensity I. And for smaller distances the intensity can be as large as $2I$.

Chapter 6

Textures & Patterns

Links

- [pov:3.8] Simple Texture Options
- [pov:3.9] Advanced Texture Options
- [pov:6.7] Textures
- [pov:10.1.10] Texture

So far we have objects and there is light, but we will still see nothing, since the objects are by default black, and hence absorb all light. We want to change this now and will give some structure to the surface (and the interior) of our objects. This is done by the following statement, which can be added as an OBJECT_MODIFIER to any object:

```
texture {
  PIGMENT_STMT
  NORMAL_STMT
  FINISH_STMT
  TRANSFORMATIONS
}
```

6.1 Pigment

Let us first discuss the PIGMENT_STMT. In its simplest form it just specifies a color C

```
pigment { color C }
```

Beside `rgb <R,G,B>` for C we can also use `rgbf <R,G,B,F>`, `rgbt <R,G,B,T>` and `rgbft <R,G,B,F,T>`.

The value F in the first variant is the filter parameter $0 \leq F \leq 1$. It specifies the amount of light of color $\langle R, G, B \rangle$ which may transmit through the object. E.g.

```
rgbf <1,0,0,0.25>
```

means that 25% of the red component of light passes through the object and nothing of the green and blue components. Thus the object acts as red filter. Note however, that the amount of filtering does not depend on the thickness of the material, see media (7.2) for how this can be made more realistic.

The value T in the second variant is the transmit parameter $0 \leq T \leq 1$. It specifies the amount of light the can pass unfiltered through the object. The remainder part $1 - T$ is used for the color $\langle R, G, B \rangle$ of the object, e.g.

```
rgbt <1,0,0,0.25>
```

means that 25% of the light will pass unfiltered though the object and 75% of red will be added to this light.

Of course, usual objects will not have a homogeneous color but some patterns, which we have to treat next.

6.1.1 Color List Patterns

These simplest patterns consist of a partitioning of space to which fixed selected colors are applied.

The first statement of this form is

```
pigment { checker color C1 color C2 }
```

This will colorize the object by a chessboard pattern of squares in the alternating colors C1 and C2. In fact all patterns in **Pov-Ray** are 3-dimensional, so this pattern colorizes that part of the space which is occupied by the object by unit cubes with alternating colors.

A modification of this theme is:

```
pigment { brick color C1 color C2 brick_size V mortar M }
```

which fills the space with bricks of dimension V in color C1 separated by gaps of dimension M in color C2. Each further higher layer of bricks is offset 1/2 in the X-direction

Another one is

```
pigment { hexagon color C1 color C2 color C3 }
```

which fills space by hexagonal infinite long upright columns of alternating colors.

6.1.2 Color Mapped Patterns

Instead of aprupt changes in colors as in the color list patterns we now want to produce colors which change gradually. For this we need a function returning float values for each point in space and some way to calculate colors out of these values.

This later step is achieved by color maps, which give a table of pairs of return values between 0 and 1 and corresponding colors. The return values of the function modulo 1 are then used to interpolate between these colors. The syntax of a color_map is:

```
color_map {
  [T1 color C1]
  [T2 color C2]
      ....
  [Tn color Cn]
}
```

Note that here the square brackets '[' and ']' belong to the syntax and are not indicators for optional parameters. The number n of colors may be between 1 and 256. The T-values should satisfy $0 \leq T_1 \leq T_2 \leq \cdots \leq T_n \leq 1$. If the function returns one of these values modulo 1 for some point in space, then the corresponding color is used. If the value T modulo 1 lies between T_i and T_{i+1} then the two colors C_i and C_{i+1} are interpolated correspondingly. If the value modulo 1 is smaller than T_1 (resp. large than T_n) then color C_1 (resp. C_n) is used.

Now we turn to the function assigning float values to all points in space. There are many predefined functions, like:

`gradient V` ... returns the length of the projection onto the vector V. Thus `gradient y` returns the height of the point for example.

`marble V` ... is similar to gradient but after reaching 1 it zig-zags the values.

`boxed` ... returns $1 - \min\{1, \max\{|X|, |Y|, |Z|\}$, which is 1 at 0 and vanishes outside the centered cube with side 2.

`cylindrical` ... returns $1 - \min\{1, \sqrt{X^2 + Z^2}\}$, which is 1 on the Y-axes and vanishes outside the cylinder with radius 1.

`onion` ... returns $\sqrt{X^2 + Y^2 + Z^2}$, which is 1 on the unitsphere and 0 at 0.

`wood` ... return $\sqrt{X^2+Z^2}$ until it reaches 1 and then it zig-zags. So this gives the rings of trees. If we want to make a board of wood, we should carefully align this texture to the box and tilt it slightly.

`bozo` ... produces a smooth random noise (= bumps and spots)

`cells` ... returns for points in each unit cube a constant random value

`crackle` ... returns 0 for points that have equal distance to the two nearest randomly selected points. This can be used for natural stone wall.

...

If these predefined functions are not sufficient the user may supply his own function as follows:

```
function { USER_FUNCTION }
```

with a user defined function USER_FUNCTION (in the simplest case this is just some expression in x, y and z), see [pov:6.1.6 User-Defined Functions]

So this type of pigment statement will look like:

```
pigment { function { ... } color_map { ... } }
```

When all else fails one may use

6.1.3 Image Maps

```
pigment {
  image_map { [TYPE] FILENAME [MODIFIER] }
}
```

Where `TYPE` is any of gif, tga, iff, ppm, pgm, png, jpeg, tiff and sys, `FILENAME` is the name of a corresponding graphic file.

The colors of the graphic file fill the unitsquare in the x-y-plane and all parallel planes repeatedly except `once` is used as MODIFIER. This way of mapping the image to the points in space may be changed by using `map_type T` instead of `TYPE`, where T can be

0 planar

1 spherical

2 cylindrical

5 toroidal

Another `MODIFIER` is `transmit all T` and `filter all F` which makes the image partially transparent by the amount T or acting as a filter by the amount F. For graphic files using a palette also `transmit IDX T` and `filter IDX F` can be used to make only pixels with color index IDX transparent.

6.2 Finish

Now we come to the FINISH_STMT:

```
finish { ... }
```

The appearance of the object does not only depend on its color pattern but also on the reflection properties of its surface. So light contributes in many ways to the appearance.

6.2.1 Ambient Light

Even if there is no light falling directly from some light source onto the object it will still be seen, since light is partially reflected or scattered by neighboring objects or even the air. This would by far be too time consuming to calculate directly by a raytracer so one simulates this ambient light by the following clause in the finish statement:

```
ambient T
```

Which says that the fraction T of the ambient light (by default of `rgb color 1`) will contribute to the color of the object. This light depends neither on the position of the object, nor the viewing angle, nor has ambient light a fixed position in space but it fills the whole space uniformly.

So the intensity it contributes to a point on the surface of an object with ambient value T is given by

$$I_a := T * C_a,$$

where C_a is the color of the ambient light.

6.2.2 Diffuse Reflected Light

Light reaching the object from some light_source will be (partially) reflected, mainly in the direction opposite to the incoming ray with respect to the surface

normal but also in all other directions in particular if the surface is rather soft like velvet.

In order to account for this later type of reflection the following clause is used in the finish statement:

```
diffuse T
```

Which says that the fraction T of the light coming (directly) from light sources will contribute to the color of the object. Note that the intensity of the light coming from the light source will also depend on the angle φ of the incoming ray with respect to the surface normal. In fact, if the amount of light reaching the surface is I_s per square unit normal to the incoming ray, then the area of the surface hit by this light is $1/\cos(\varphi)$ and hence the intensity it contributes is

$$I_d := T * I_s * \cos(\varphi).$$

6.2.3 Brilliance

Values of brilliance higher than the default 1.0 makes the light fall off less at medium to small angles thus making the object to appear more metallic and of higher surface shininess.

6.2.4 Highlights

These are bright spots which appear when light coming directly from a light source reflects off a smooth surface. This can be simulated by

```
finish { ... phong P phong_size S [metallic] ... }
```

where $0 \leq P \leq 1$ specifies the amount by which the light is saturated by the surface color (1 means complete saturation). The phong_size S is inverse proportional to the size of the highlight.

The formula for the corresponding light intensity is

$$I_p := \frac{1}{S} * I_d * \cos(\theta)^n,$$

for some exponent n.

An alternative method (which is better on low viewing angles) is

```
finish { ... specular P roughness R [metallic] ... }
```

where $0 \leq P \leq 1$ specifies the amount by which the light is saturated by the surface color. (1 is complete saturation). The roughness R is proportional to the size of the highlight.

The optional keyword `metallic` gives the object a more metallic appearance.

6.2.5 Specular Reflection

The part of the light directly coming from light sources but also indirectly via other objects and which is mirrored by the surface is controlled by the following part of in the finish statement:

```
reflection { color C [, color C1] [exponent E]
  [falloff F] [metallic M] [fresnel B] [conserve_energy]
}
```

where the part C of the incoming light is reflected, `falloff` changes the behavior in such a way that darker objects are reflected less (which corresponds better to reality), usage of `color C1` (e.g. for water) makes the reflection depending on the incoming=outgoing angle (i.e. C is used for normal hitting light, $C1$ for light parallel to the surface) and the interpolation inbetween is controlled by `exponent E` (for linear ($E = 1$), quadratic ($E = 2$), ...). The option `conserve_energy` adjusts the amount of light transmitted and filtered by the amount reflected. Using the phrase `metallic` means that the reflected light is multiplied by the surface color.

6.2.6 Refraction

When light passes through an object (having non-zero transmit or filter parameter) of different density then the surrounding air the light is bended. This is called refraction. By default transparent objects in **Pov-Ray** do not refract light. But this can be turned on by

```
finish { ... refraction 1 ior K ... }
```

where K is the i(ndex)o(f)r(refraction). Water has an ior of 1.33, glass of approximately 1.5 and diamond of 2.4 (see "ior.inc").

6.3 Normal

Next we come to the NORMAL_STMT. Note that light given by `diffuse`, highlights, `reflection` and `refraction` depends on the normal to the surface of the

object. For some objects (like planes) the surface normal is explicitly given, for other it is obvious (like boxes (except at the edges), spheres, tori, etc.). It is not for all objects an easy problem to calculate the surface normal. For an implicitly given (iso)surface $M = f^{-1}(0)$ via a C^1-function f, the normal to M in a point $x \in M$ is described by the gradient

$$\operatorname{grad} f(x_1, x_2, x_3) = \left\langle \frac{\partial}{\partial x_1} f(x_1, x_2, x_3), \frac{\partial}{\partial x_2} f(x_1, x_2, x_3), \frac{\partial}{\partial x_3} f(x_1, x_2, x_3) \right\rangle$$

provided this does not vanish: In fact, the gradient of f is related to the derivative by

$$\langle \operatorname{grad} f(x_1, x_2, x_3) | (v_1, v_2, v_3) \rangle = \sum_{i=1}^{3} v_i \cdot \frac{\partial f}{\partial x_i}(x_1, x_2, x_3) = f'(x_1, x_2, x_3) \cdot (v_1, v_2, v_3)$$

That a vector $v = (v_1, v_2, v_3)$ stands normal to the surface M at the point $P := (x_1, x_2, x_3)$ means that for any differentiable curve c in M with $c(0) = P$ we have $\langle v | c'(0) \rangle = 0$. That c lies in $M = f^{-1}(0)$ means that $f \circ c$ is constant and hence

$$0 = (f \circ c)'(0) = f'(c(0)) \cdot c'(0) = \langle \operatorname{grad} f(c(0)) | c'(0) \rangle,$$

by the chain-rule.

Furthermore, natural objects usually don't have mathematically perfect surfaces but irregularities like bumps, cracks or some graininess, and it would be very time-consuming to model them. Cheaper is to modify only the (virtual) normal to the object. This can be done in various ways inside the texture statement by the normal statement:

```
NORMAL:
  normal { [NORMAL_IDENTIFIER] [NORMAL_TYPE] [NORMAL_MODIFIER...] }

NORMAL_TYPE:
  PATTERN_TYPE [Amount] |
  bump_map { BITMAP_TYPE "bitmap.ext" [BUMP_MAP_MODS...]}

NORMAL_MODIFIER:
  PATTERN_MODIFIER |
NORMAL_LIST |
  slope_map { SLOPE_MAP_BODY } |
normal_map { NORMAL_MAP_BODY } |
bump_size Amount |
  no_bump_scale Bool | accuracy Float
```

There are four basic NORMAL_TYPEs. They are:

- block pattern normals,
- continuous pattern normals,
- specialized normals,
- bump maps.

The pattern type is optionally followed by one or more normal modifiers. Different modifiers may be used with the various NORMAL_TYPEs. In addition to general pattern modifiers such as transformations, turbulence, and warp modifiers, normals may also have a NORMAL_LIST, slope_map, normal_map, and bump_size which are specific to normals. Normal modifiers of any kind apply only to the normal and not to other parts of the texture.

6.3.1 Slope Map

> From [pov:6.7.2.1] :
> Each of the various pattern types available is in fact a mathematical function that takes any x, y, z location and turns it into a number between 0.0 and 1.0 inclusive. That number is used to specify where the various high and low spots are. The slope_map lets you further shape the contours.

This is like in a color_map: We have some function and a table called `slope_map`:

```
normal {
  FUNCTION
  slope_map {
    [ T1, <H1,K1> ]
    [ T2, <H2,K2> ]
       ...
    [ TN, <HN,KN> ]
  }
}
```

where $T1$, $T2$,...,Tn are the special return values of the function and the H's and K's are the corresponding heights and slopes. The slopes inbetween are interpolated by cubic splines.

6.3.2 Normal Map

We can even blend various types of normals using a FUNCTION and a `normal_map`.

6.3.3 Bump Map

From [pov:6.7.2.3] :
Instead of placing the color of the image on the shape like an image_map a bump_map perturbs the surface normal based on the color of the image at that point. The result looks like the image has been embossed into the surface. By default, a bump map uses the brightness of the actual color of the pixel.

```
BUMP_MAP:
    normal
    {
        bump_map
        {
            BITMAP_TYPE "bitmap.ext"
            [BUMP_MAP_MODS...]
        }
        [NORMAL_MODFIERS...]
    }

BITMAP_TYPE:
    gif | tga | iff | ppm | pgm | png | jpeg | tiff | sys

BUMP_MAP_MOD:
    map_type Type | once | interpolate Type | use_color |
    use_colour | bump_size Value
```

6.3.4 Patterned Textures

From [pov:6.7.5] :
Patterned textures are complex textures made up of multiple textures. The component textures may be plain textures or may be made up of patterned textures. A plain texture has just one pigment, normal and finish statement. Even a pigment with a pigment map is still one pigment and thus considered a plain texture as are normals with normal map statements.

Patterned textures use either a texture_map statement to specify a blend or pattern of textures or they use block textures such as checker with a texture

list or a bitmap similar to an image map called a material map specified with a material_map statement.

An example using a texture map is:

```
texture {
  gradient x       // this is the PATTERN_TYPE
  texture_map {
    [0.3  pigment{Red} finish{phong 1}]
    [0.3  T_Wood11]    // this is a texture identifier
    [0.6  T_Wood11]
    [0.9  pigment{DMFWood4} finish{Shiny}]
  }
}
```

Material Map

Instead of placing a solid color of the image on the shape like an image map, an entire texture is specified based on the index or color of the image at that point. You must specify a list of textures to be used like a texture palette rather than the usual color palette.

```
MATERIAL_MAP:
    texture
    {
        material_map
        {
            BITMAP_TYPE "bitmap.ext"
            [BITMAP_MODS...] TEXTURE... [TRANSFORMATIONS...]
        }
    }
BITMAP_TYPE:
    gif | tga | iff | ppm | pgm | png | jpeg | tiff | sys
BITMAP_MOD:
    map_type Type | once | interpolate Type
```

Layered Texture:

A layered texture consists of several textures that are partially transparent and are laid one on top of the other to create a more complex texture. The different texture layers show through the transparent portions to create the appearance of one texture that is a combination of several textures.

You create layered textures by listing two or more textures one right after the other. The last texture listed will be the top layer, the first one listed will be the bottom layer. All textures in a layered texture other than the bottom layer should have some transparency. For example:

```
object {
  My_Object
  texture {T1}  // the bottom layer
  texture {T2}  // a semi-transparent layer
  texture {T3}  // the top semi-transparent layer
}
```

In this example T2 shows only where T3 is transparent and T1 shows only where T2 and T3 are transparent.

6.3.5 UV-Mappings

From [pov:6.7.7] :
All textures in POV-Ray are defined in 3 dimensions. Even planar image mapping is done this way. However, it is sometimes more desirable to have the texture defined for the surface of the object. This is especially true for bicubic_patch objects and mesh objects, that can be stretched and compressed. When the object is stretched or compressed, it would be nice for the texture to be glued to the object's surface and follow the object's deformations.

When uv_mapping is used, then that object's texture will be mapped to it using surface coordinates (u and v) instead of spatial coordinates (x, y, and z). This is done by taking a slice of the object's regular 3D texture from the XY plane (Z=0) and wrapping it around the surface of the object, following the object's surface coordinates.

Note: some textures should be rotated to fit the slice in the XY plane.

Syntax:

```
texture {
 pigment { uv_mapping PIGMENT_BODY }
 normal  { uv_mapping NORMAL_BODY  }
 texture { uv_mapping TEXTURE_BODY }
}
```

Surface mapping is currently defined for the following objects:

bicubic_patch UV coordinates are based on the patch's parametric coordinates. They stretch with the control points. The default range is (0..1) and can be changed.

mesh, mesh2 UV coordinates are defined for each vertex and interpolated between.

lathe, sor modified spherical mapping... the u coordinate (0..1) wraps around the y axis, while the v coordinate is linked to the object's control points (also ranging 0..1). Surface of Revolution also has special disc mapping on the end caps if the object is not 'open'.

sphere boring spherical mapping.

box the image is wrapped around the box.

With the keyword `uv_vectors`, the UV coordinates of the corners can be controlled for bicubic patches and standard triangle mesh.

For bicubic patches the UV coordinates can be specified for each of the four corners of the patch. This goes right before the control points. The syntax is:

```
uv_vectors <corner1>,<corner2>,<corner3>, <corner4>
```

with default

```
uv_vectors <0,0>,<1,0>,<1,1>,<0,1>
```

Similarly for mesh it looks like

```
mesh {
  triangle { ... }
uv_vectors <0.0>,<1,0>,<1,1>
...
texture { pigment { uv_mapping image_map ... } }
}
```

Interior Texture

From [pov:6.7.9] :
Syntax:

```
object {
  texture { TEXTURE_ITEMS... }
  interior_texture { TEXTURE_ITEMS...}
}
```

All surfaces have an exterior and interior surface. The interior_texture simply allows to specify a separate texture for the interior surface of the object. For objects with no well defined inside/outside (bicubic_patch, triangle, ...) the interior_texture is applied to the backside of the surface. Interior surface textures use exactly the same syntax and should work in exactly the same way as regular surface textures, except that they use the keyword `interior_texture` instead of `texture`.

Note: Do not confuse interior_texture {} with interior {}: the first one specifies surface properties, the second one specifies volume properties.

6.4 Pattern Modifier

There are various ways to modify a pattern, we will treat only the most important ones.

6.4.1 Transforming Patterns

From [pov:6.7.12.1] :
The most common pattern modifiers are the transformation modifiers translate, rotate, scale, transform, and matrix, see chapter (4). These modifiers may be placed inside pigment, normal, texture, and density statements to change the position, size and orientation of the patterns.

6.4.6 Warps

Warps replace points on the object by some points nearby specified in various ways.

From [pov:6.7.12.6] :
Currently there are seven types of warps but the syntax was designed to allow future expansion.

The syntax for using a warp statement is:

```
WARP:
    warp {
      repeat <Direction> [REPEAT_ITEMS...] |
      black_hole <Location>, Radius [BLACK_HOLE_ITEMS...] |
      turbulence <Amount> [TURB_ITEMS...]
      cylindrical  [ orientation VECTOR | dist_exp FLOAT ]
      spherical  [ orientation VECTOR | dist_exp FLOAT ]
      toroidal  [ orientation VECTOR | dist_exp FLOAT |
                major_radius FLOAT ]
      planar [ VECTOR , FLOAT ]
}

REPEAT_ITEMS:
    offset <Amount> |
    flip <Axis>

BLACK_HOLE_ITEMS:
    strength Strength | falloff Amount | inverse |
    repeat <Repeat> | turbulence <Amount>

TURB_ITEMS:
    octaves Count | omega Amount | lambda Amount
```

6.4.7 Black Hole Warp

From [pov:6.7.12.6.1] :

```
BLACK_HOLE_WARP:
    warp
    {
        black_hole <Location>, Radius
        [BLACK_HOLE_ITEMS...]
    }
BLACK_HOLE_ITEMS:
    strength Strength | falloff Amount | inverse | type Type |
    repeat <Repeat> | turbulence <Amount>
```

The minimal requirement is the `black_hole` keyword followed by a vector ⟨Location⟩ followed by a comma and a float Radius. Black holes effect all points

within the spherical region around the location and within the radius. This is optionally followed by any number of other keywords which control how the texture is warped.

The amount of displacement of the point is given by the formula

$$\text{strength} * d^{\text{falloff}},$$

where d is the distance of the point from the complement of the ball around $\langle Location \rangle$ with radius 1, so $d = 0$ outside this ball and $d = 1$ on its center.

The keyword `inverse` reverses the direction of the displacement.

The phrase `repeat V` makes black holes at the lattice points specified by the components of V.

6.4.8 Repeat Warp

From [pov:6.7.12.6.2] :
The repeat warp causes a section of the pattern to be repeated over and over. It takes a slice out of the pattern and makes multiple copies of it side-by-side. The warp has many uses but was originally designed to make it easy to model wood veneer textures. Veneer is made by taking very thin slices from a log and placing them side-by-side on some other backing material. You see side-by-side nearly identical ring patterns but each will be a slice perhaps 1/32th of an inch deeper.

The syntax for a repeat warp is

```
REPEAT_WARP:
    warp { repeat <Direction> [REPEAT_ITEMS...] }

REPEAT_ITEMS:
    offset <Amount> | flip <Axis>
```

The repeat vector specifies the direction in which the pattern repeats and the width of the repeated area. This vector must lie entirely along an axis. In other words, two of its three components must be 0. For example

```
pigment {
  wood
  warp { repeat 2*x }
}
```

which means that from x=0 to x=2 you get whatever the pattern usually is. But from x=2 to x=4 you get the same thing exactly shifted two units over in the x-direction. To evaluate it you simply take the x-coordinate modulo 2. Unfortunately you get exact duplicates which isn't very realistic. The optional offset vector tells how much to translate the pattern each time it repeats. For example

```
pigment {
    wood
    warp {repeat x*2  offset z*0.05}
  }
```

means that we slice the first copy from x=0 to x=2 at z=0 but at x=2 to x=4 we offset to z=0.05. In the 4 to 6 interval we slice at z=0.10. At the n-th copy we slice at 0.05 n z. Thus each copy is slightly different. There are no restrictions on the offset vector.

Finally the flip vector causes the pattern to be flipped or mirrored every other copy of the pattern. The first copy of the pattern in the positive direction from the axis is not flipped. The next farther is, the next is not, etc. The flip vector is a three component x, y, z vector but each component is treated as a boolean value that tells if you should or should not flip along a given axis. For example

```
pigment {
    wood
    warp {repeat 2*x  flip <1,1,0>}
}
```

means that every other copy of the pattern will be mirrored about the x- and y-axis but not the z-axis. A non-zero value means flip and zero means do not flip about that axis. The magnitude of the values in the flip vector doesn't matter.

6.4.9 Turbulence Warp

From [pov:6.7.12.6.4] :
Inside the warp statement, the keyword `turbulence` followed by a float or vector may be used to stir up any pigment, normal or density. A number of optional parameters may be used with turbulence to control how it is computed. The syntax is:

```
TURBULENCE_ITEM:
    turbulence <Amount> | octaves Count |
    omega Amount | lambda Amount
```

Typical turbulence values range from the default 0.0, which is no turbulence, to 1.0 or more, which is very turbulent. If a vector is specified different amounts of turbulence are applied in the x, y and z-direction.

Turbulence uses a random noise function called DNoise. This is similar to the noise used in the bozo pattern except that instead of giving a single value it gives a direction. You can think of it as the direction that the wind is blowing at that spot. Points close together generate almost the same value but points far apart are randomly different.

Turbulence uses DNoise to push a point around in several steps called octaves. We locate the point we want to evaluate, then push it around a bit using turbulence to get to a different point then look up the color or pattern of the new point.

It says in effect "Don't give me the color at this spot... take a few random steps in different directions and give me that color". Each step is typically half as long as the one before.

From [pov:6.7.12.6.4.1] :

Octaves

The `octaves` keyword may be followed by an integer value to control the number of steps of turbulence that are computed. Legal values range from 1 to ⟨10. The default value of 6 is a fairly high value; you won't see much change by setting it to a higher value because the extra steps are too small. Float values are truncated to integer. Smaller numbers of octaves give a gentler, wavy turbulence and computes faster. Higher octaves create more jagged or fuzzy turbulence and takes longer to compute. Lambda

The `lambda` parameter controls how statistically different the random move of an octave is compared to its previous octave. The default value is 2.0 which is quite random. Values close to lambda 1.0 will straighten out the randomness of the path in the diagram above. The zig-zag steps in the calculation are in nearly the same direction. Higher values can look more swirly under some circumstances.

Omega

The omega value controls how large each successive octave step is compared to the previous value. Each successive octave of turbulence is multiplied by the omega value. The default omega 0.5 means that each octave is 1/2 the size of

the previous one. Higher omega values mean that 2nd, 3rd, 4th and up octaves contribute more turbulence giving a sharper, crinkly look while smaller omegas give a fuzzy kind of turbulence that gets blurry in places.

Chapter 7

Media & Atmosphere

7.1 Interior

7.1.1 Empty and Solid Objects

From [pov:6.6.2] :
It is very important that you know the basic concept behind empty and solid objects in POV-Ray to fully understand how features like interior and translucency are used. Objects in POV-Ray can either be solid, empty or filled with (small) particles.

A solid object is made from the material specified by its pigment and finish statements (and to some degree its normal statement). By default all objects are assumed to be solid. If you assign a stone texture to a sphere you'll get a ball made completely of stone. It's like you had cut this ball from a block of stone. A glass ball is a massive sphere made of glass. You should be aware that solid objects are conceptual things. If you clip away parts of the sphere you'll clearly see that the interior is empty and it just has a very thin surface.

This is not contrary to the concept of a solid object used in POV-Ray. It is assumed that all space inside the sphere is covered by the sphere's interior. Light passing through the object is affected by attenuation and refraction properties. However there is no room for any other particles like those used by fog or interior media.

Empty objects are created by adding the `hollow` keyword (see "Hollow") to the object statement. An empty (or hollow) object is assumed to be made of a very thin surface which is of the material specified by the pigment, finish and normal statements. The object's interior is empty, it normally contains air molecules.

An empty object can be filled with particles by adding fog or atmospheric media to the scene or by adding an interior media to the object. It is very important to understand that in order to fill an object with any kind of particles it first has to be made hollow.

7.1.2 The Interior Statement

From [pov:6.6] :

```
INTERIOR:
    interior { [INTERIOR_IDENTIFIER] [INTERIOR_ITEMS...] }
INTERIOR_ITEM:
    ior Value | caustics Value | dispersion Value |
    dispersion_samples Samples | fade_distance Distance |
    fade_power Power | fade_color <Color>
    MEDIA...

Interior default values:

ior                : 1.0
caustics           : 0.0
dispersion         : 1.0
dispersion_samples : 7
fade_distance      : 0.0
fade_power         : 0.0
fade_color         : <0,0,0>
```

7.1.3 Refraction

From [pov:6.6.4] :
When light passes through a surface either into or out of a dense medium the path of the ray of light is bent. Such bending is called refraction. The amount of bending or refracting of light depends upon the density of the material. Air, water, crystal and diamonds all have different densities and thus refract differently. The index of refraction or ior value is used by scientists to describe the relative density of substances. The `ior` keyword is used in POV-Ray in the interior to turn on refraction and to specify the `ior` value.

7.1.4 Dispersion

From [pov:6.6.5] :
For all materials with a ior different from 1.0 the refractive index isn't constant throughout the spectrum. It changes as a function of wavelength. Generally the refractive index decreases as the wavelength increases. Therefore light passing through a material will be separated according to wavelength. This is known as chromatic dispersion.

By default POV-Ray does not calculate dispersion as light travels through a transparent object. In order to get a more realistic effect the `dispersion` and `dispersion_samples` keywords can be added to the interior block. They will simulate dispersion by creating a prismatic color effect in the object.

The dispersion value is the ratio of refractive indices for violet to red. It controls the strength of dispersion (how much the colors are spread out) used. A DISPERSION_VALUE of 1 will give no dispersion, good values are 1.01 to 1.1.

7.1.5 Attenuation

From [pov:6.6.6] :
Light attenuation is used to model the decrease in light intensity as the light travels through a transparent object. The keywords `fade_power`, `fade_distance` and `fade_color` are specified in the interior statement.

The `fade_distance` value determines the distance the light has to travel to reach half intensity while the `fade_power` value determines how fast the light will fall off. `fade_color` colorizes the attenuation. For realistic effects a fade power of 1 to 2 should be used. Default values for `fade_power` and `fade_distance` is 0.0 which turns this feature off. Default for `fade_color` is ⟨0,0,0⟩, if `fade_color` is ⟨1,1,1⟩ there is no attenuation. The actual colors give colored attenuation. ⟨1,0,0⟩ looks red, not cyan as in media.

The attenuation is calculated by a formula similar to that used for light source attenuation.

$$\text{attenuation} = \frac{1}{1 + (\frac{d}{\text{fade\_distance}})^{\text{fade\_power}}}$$

If you set `fade_power` in the interior of an object at 1000 or above, a realistic exponential attenuation function will be used:

$$\text{attenuation} = \exp(-\,\text{depth} \,/\, \text{fade\_dist})$$

7.1.6 Object-Media

From [pov:6.6.8] :
The interior statement may contain one or more media statements. Media is used to simulate suspended particles such as smoke, haze, or dust. Or visible gasses such as steam or fire and explosions. When used with an object interior, the effect is constrained by the object's shape. The calculations begin when the ray enters an object and ends when it leaves the object. Any interior media patterns are totally independent of the texture.

7.2 Media

From [pov:6.8] :
The media statement is used to specify particulate matter suspended in a medium such air or water. It can be used to specify smoke, haze, fog, gas, fire, dust etc.

```
MEDIA:
    media { [MEDIA_IDENTIFIER] [MEDIA_ITEMS...] }
MEDIA_ITEMS:
    method Number | intervals Number | samples Min, Max |
    confidence Value  | variance Value | ratio Value |
    absorption COLOR | emission COLOR | aa_threshold Value |
    aa_level Value |
    scattering {
       Type, COLOR [ eccentricity Value ] [ extinction Value ]
    }  |
    density {
       [DENSITY_IDENTIFIER] [PATTERN_TYPE]
       [DENSITY_MODIFIER...]
    }   |
    TRANSFORMATIONS
DENSITY_MODIFIER:
    PATTERN_MODIFIER | DENSITY_LIST | COLOR_LIST
    | color_map { COLOR_MAP_BODY }
    | colour_map { COLOR_MAP_BODY }
    | density_map { DENSITY_MAP_BODY }

Media default values:

  aa_level     : 4
```

```
aa_threshold : 0.1
absorption   : <0,0,0>
confidence   : 0.9
emission     : <0,0,0>
intervals    : 10
method       : 3
ratio        : 0.9
samples      : Min 1, Max 1
variance     : 1/128
SCATTERING COLOR : <0,0,0>
eccentricity : 0.0
extinction   : 1.0
```

7.2.1 Media Types

From [pov:6.8.1] :
There are three types of particle interaction in media: absorbing, emitting, and scattering. All three activities may occur in a single media. Each of these three specifications requires a color. Only the red, green, and blue components of the color are used.

7.2.2 Density

From [pov:6.8.3] :
Particles of media are normally distributed in constant density throughout the media. However the density statement allows you to vary the density across space using any of POV-Ray's pattern functions such as those used in textures. If no density statement is given then the density remains a constant value of 1.0 throughout the media. More than one density may be specified per media statement. The syntax for density is:

```
DENSITY:
    density
    {
        [DENSITY_IDENTIFIER]
        [DENSITY_TYPE]
        [DENSITY_MODIFIER...]
    }
DENSITY_TYPE:
```

```
    PATTERN_TYPE | COLOR
DENSITY_MODIFIER:
    PATTERN_MODIFIER | DENSITY_LIST
    | color_map { COLOR_MAP_BODY }
    | colour_map { COLOR_MAP_BODY }
    | density_map { DENSITY_MAP_BODY }
```

The density statement may begin with an optional density identifier. All subsequent values modify the defaults or the values in the identifier. The next item is a pattern type. This is any one of POV-Ray's pattern functions such as bozo, wood, gradient, waves, etc. Of particular usefulness are the spherical, planar, cylindrical, and boxed patterns which were previously available only for use with our discontinued halo feature. All patterns return a value from 0.0 to 1.0. This value is interpreted as the density of the media at that particular point. See "Patterns" for details on particular pattern types. Although a solid COLOR pattern is legal, in general it is used only when the density statement is inside a density_map.

7.3 Atmospheric Effects

From [pov:6.9] :
Atmospheric effects such as fog, dust, haze, or visible gas may be simulated by a media statement specified in the scene but not attached to any object.

7.3.1 Background

From [pov:6.9.2] :
A background color can be specified if desired. Any ray that doesn't hit an object will be colored with this color.

7.3.2 Sky Sphere

From [pov:6.9.4] :
The sky sphere is used create a realistic sky background without the need of an additional sphere to simulate the sky. Its syntax is:

```
SKY_SPHERE:
    sky_sphere { [SKY_SPHERE_IDENTIFIER] [SKY_SPHERE_ITEMS...] }
```

```
SKY_SPHERE_ITEM:
    PIGMENT | TRANSFORMATION
```

The sky sphere can contain several pigment layers with the last pigment being at the top, i.e. it is evaluated last, and the first pigment being at the bottom, i.e. it is evaluated first. If the upper layers contain filtering and/or transmitting components lower layers will shine through. If not lower layers will be invisible.

The sky sphere is calculated by using the direction vector as the parameter for evaluating the pigment patterns. This leads to results independent from the view point which pretty good models a real sky where the distance to the sky is much larger than the distances between visible objects.

If you want to add a nice color blend to your background you can easily do this by using the following example.

```
sky_sphere {
    pigment {
      gradient y
      color_map {
        [ 0.5  color CornflowerBlue ]
        [ 1.0  color MidnightBlue ]
      }
      scale 2
      translate -1
    }
  }
```

This gives a soft blend from CornflowerBlue at the horizon to MidnightBlue at the zenith. The scale and translate operations are used to map the direction vector values, which lie in the range from ⟨-1, -1, -1⟩ to ⟨1, 1, 1⟩, onto the range from ⟨0, 0, 0⟩ to ⟨1, 1, 1⟩. Thus a repetition of the color blend is avoided for parts of the sky below the horizon.

7.3.3 Rainbow

From [pov:6.9.5] :

```
RAINBOW:
    rainbow { [RAINBOW_IDENTIFIER] [RAINBOW_ITEMS...] }
```

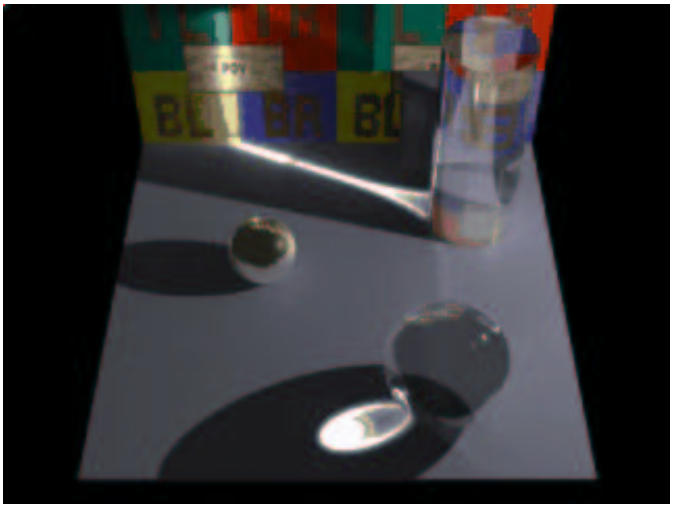
```
RAINBOW_ITEM:
    direction <Dir> | angle Angle | width Width |
    distance Distance | COLOR_MAP | jitter Jitter | up <Up> |
    arc_angle Arc_Angle | falloff_angle Falloff_Angle

Rainbow default values:

arc_angle     : 180.0
falloff_angle : 180.0
jitter        : 0.0
up            : y
```

7.4 Photons

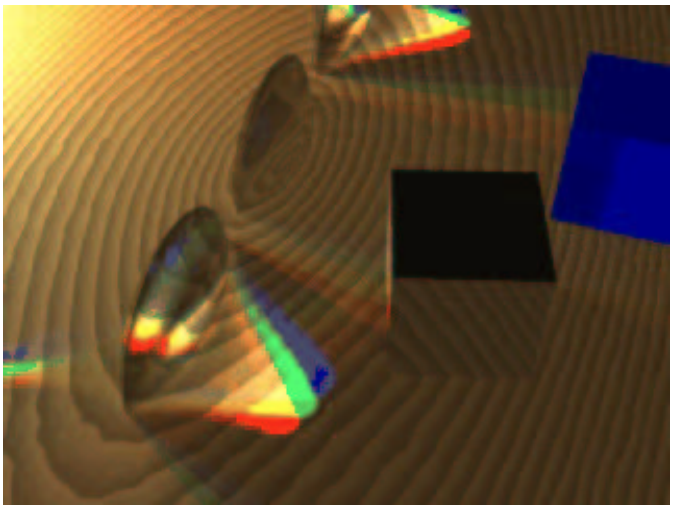
From [pov:6.10] :
The basic goal of this implementation of the photon map is to render true reflective and refractive caustics. The photon map was first introduced by Henrik Wann Jensen.

Photon mapping is a technique which uses a backwards ray-tracing pre-processing step to render refractive and reflective caustics realistically. This means that mirrors can reflect light rays and lenses can focus light.

Photon mapping works by shooting packets of light (photons) from light sources into the scene. The photons are directed towards specific objects. When a photon hits an object after passing through (or bouncing off of) the target object, the ray intersection is stored in memory. This data is later used to estimate the amount of light contributed by reflective and refractive caustics.



Example using photons



Another example using photons

7.4.1 Using Photon Mapping in Your Scene

From [pov:6.10.2] :
When designing a scene with photons, it helps to think of the scene objects in two categories. Objects in the first category will show photon caustics when hit by photons. Objects in the second category cause photon caustics by reflecting or refracting photons. Some objects may be in both categories, and some objects may be in neither category.

Category 1 - Objects that show photon caustics

By default, all objects are in the first category. Whenever a photon hits an object, the photon is stored and will later be used to render caustics on that object. This means that, by default, caustics from photons can appear on any surface. To speed up rendering, you can take objects out of this category. You do this with the line: `photons { collect off }`. If you use this syntax, caustics from photons will not appear on the object. This will save both memory and computational time during rendering.

Category 2 - Objects that cause photon caustics

By default, there are no objects in the second category. If you want your object to cause caustics, you need to do two things. First, make your object into a "target." You do this with the `target` keyword. This enables light sources to shoot photons at your object. Second, you need to specify if your object reflects photons, refracts photons, or both. This is done with the `reflection on` and `refraction on` keywords. To allow an object to reflect and refract photons, you would use the following lines of code inside the object:

```
photons{
  target
  reflection on
  refraction on
}
```

Generally speaking, you don't want an object to be in both categories. Most objects that cause photon caustics do not themselves have much color or brightness. Usually they simply refract or reflect their surroundings. For this reason, it is usually a waste of time to display photon caustics on such surfaces. Even if computed, the effects from the caustics would be so dim that they would go unnoticed.

Sometimes, you may also wish to add photons{collect off} to other clear or reflective objects, even if they are not photon targets. Again, this is done to prevent unnecessary computation of caustic lighting.

Finally, you may wish to enable photon reflection and refraction for a surface, even if it is not a target. This allows indirect photons (photons that have already hit a target and been reflected or refracted) to continue their journey after hitting this object.

7.4.2 Photon Global Settings

From [pov:6.10.2.1] :

```
global_photon_block:
photons {
  spacing <photon_spacing> | count <photons_to_shoot>

  [gather <min_gather>, <max_gather>]
  [media <max_steps> [,<factor>]]
  [jitter <jitter_amount>]
  [max_trace_level <photon_trace_level>]
  [adc_bailout <photon_adc_bailout>]
  [save_file "filename" | load_file "filename"]
  [autostop <autostop_fraction>]
  [expand_thresholds <percent_increase>, <expand_min>]
  [radius <gather_radius>,<multiplier>,
          <gather_radius_media>,<multiplier>]
}

All photons default values:

Global :
  expand_min    : 40
  gather        : 20, 100
  jitter        : 0.4
  media         : 0

Object :
  collect       : on
  refraction    : off
  reflection    : off
  split_union   : on
  target        : 1.0

Light_source:
  area_light    : off
  refraction    : off
  reflection    : off
```

The number of photons generated can be set using either the spacing or count keywords:

- If spacing is used, it specifies approximately the average distance between photons on surfaces. If you cut the spacing in half, you will get four times

as many surface photons, and eight times as many media photons.

- If count is used, POV-Ray will shoot the approximately number of photons specified. The actual number of photons that result from this will almost always be at least slightly different from the number specified. Still, if you double the photons_to_shoot value, then twice as many photons will be shot. If you cut the value in half, then half the number of photons will be shot.
- – It may be less, because **Pov-Ray** shoots photons at a target object's bounding box, which means that some photons will miss the target object.
  – On the other hand, may be more, because each time one object hits an object that has both reflection and refraction, two photons are created (one for reflection and one for refraction).
  – POV will attempt to compensate for these two factors, but it can only estimate how many photons will actually be generated. Sometimes this estimation is rather poor, but the feature is still usable.

The keyword `gather` allows you to specify how many photons are gathered at each point during the regular rendering step. The first number (default 20) is the minimum number to gather, while the second number (default 100) is the maximum number to gather. These are good values and you should only use different ones if you know what you're doing.

The keyword `media` turns on media photons. The parameter `max_steps` specifies the maximum number of photons to deposit over an interval. The optional parameter `factor` specifies the difference in media spacing compared to surface spacing. You can increase `factor` and decrease `max_steps` if too many photons are being deposited in media.

The keyword `jitter` specifies the amount of jitter used in the sampling of light rays in the pre-processing step. The default value is good and usually does not need to be changed.

The keywords `max_trace_level` and `adc_bailout` allow you to specify these attributes for the photon-tracing step. If you do not specify these, the values for the primary ray-tracing step will be used.

The keywords `save_file` and `load_file` allow you to save and load photon maps. If you load a photon map, no photons will be shot. The photon map file contains all surface (caustic) and media photons.

The keyword `radius` is used for gathering photons. The larger the radius, the longer it takes to gather photons. But if you use too small of a radius, you might

not get enough photons to get a good estimate. Therefore, choosing a good radius is important. Normally POV-Ray looks through the photon map and uses some ad-hoc statistical analysis to determine a reasonable radius. Sometimes it does a good job, sometimes it does not. The `radius` keyword lets you override or adjust POV-Ray's guess.

The keywords `autostop` and `expand_thresholds` will be explained later.

From [pov:6.10.2.2] :

Shooting Photons at an Object

```
object_photon_block:
photons {
  [target [<spacing_multiplier>]]
  [refraction on|off]
  [reflection on|off]
  [collect on|off]
  [pass_through]
}
```

To shoot photons at an object, you need to tell **Pov-Ray** that the object receives photons. To do this, create a photons { } block within the object. For example:

```
object {
  MyObject
  photons {
    target
    refraction on
    reflection on
    collect off
  }
}
```

In this example, the object both reflects and refracts photons. Either of these options could be turned off (by specifying reflection off, for example). By using this, you can have an object with a reflective finish which does not reflect photons for speed and memory reasons.

The keyword `target` makes this object a target.

The density of the photons can be adjusted by specifying the spacing_multiplier. If, for example, you specify a spacing_multiplier of 0.5, then the spacing for photons hitting this object will be 1/2 of the distance of the spacing for other objects.

Note: This means four times as many surface photons, and eight times as many media photons.

The keyword `collect off` causes the object to ignore photons. Photons are neither deposited nor gathered on that object.

The keyword `pass_through` causes photons to pass through the object unaffected on their way to a target object. Once a photon hits the target object, it will ignore the pass_through flag. This is basically a photon version of the `no_shadow` keyword, with the exception that media within the object will still be affected by the photons (unless that media specifies collect off). If you use the `no_shadow` keyword, the object will be tagged as pass_through automatically. You can then turn off pass_through if necessary by simply using `photons { pass_through off }`.

Note: Photons will not be shot at an object unless you specify the `target` keyword. Simply turning refraction on will not suffice.

When shooting photons at a CSG-union, it may sometimes be of advantage to use split_union off inside the union. POV-Ray will be forced to shoot at the whole object, instead of splitting it up and shooting photons at its compound parts.

From [pov:6.10.2.3] :

Photons and Light Sources

```
light_photon_block:
photons {
  [refraction on | off]
  [reflection on | off]
  [area_light]
}

Example:

light_source {
  MyLight
  photons {
    refraction on
```

```
    reflection on
  }
}
```

Sometimes, you want photons to be shot from one light source and not another. In that case, you can turn photons on for an object, but specify
`photons { reflection off refraction off }`
in the light source's definition. You can also turn off only reflection or only refraction for any light source.

See also:
- [pov:6.10.3] Photons FAQ
- [pov:6.10.4] Photon Tips
- [pov:6.10.5] Advanced Techniques

7.5 Radiosity

From [pov:4.1.1] :

7.5.1 Introduction

Radiosity is a lighting technique to simulate the diffuse exchange of radiation between the objects of a scene. With a raytracer like POV-Ray, normally only the direct influence of light sources on the objects can be calculated, all shadowed parts look totally flat. Radiosity can help to overcome this limitation. More details on the technical aspects can be found in the reference section.

To enable radiosity, you have to add a radiosity block to the global_settings in your POV-Ray scene file. Radiosity is more accurate than simplistic ambient light but it takes much longer to compute, so it can be usefull to switch off radiosity during scene development. You can use a declared constant or an INI-file constant and an #if statement to do this:

```
#declare RAD = off;

  global_settings {
     #if(RAD)
        radiosity {
```

```
            ...
        }
    #end
}
```

Most important for radiosity are the ambient and diffuse finish components of the objects. Their effect differs quite much from a conventionally lit scene.

- ambient: specifies the amount of light emitted by the object. This is the basis for radiosity without conventional lighting but also in scenes with light sources this can be important. Since most materials do not actually emit light, the default value of 0.1 is too high in most cases. You can also change ambient_light to influence this.
- diffuse: influences the amount of diffuse reflection of incoming light. In a radiosity scene this does not only mean the direct appearance of the surface but also how much other objects are illuminated by indirect light from this surface.

From [pov:6.11.11] :
Important notice: The radiosity features in POV-Ray are somewhat experimental. There is a high probability that the design and implementation of these features will be changed in future versions. We cannot guarantee that scenes using these features in this version will render identically in future releases or that full backwards compatibility of language syntax can be maintained.

Radiosity is an extra calculation that more realistically computes the diffuse interreflection of light. This diffuse interreflection can be seen if you place a white chair in a room full of blue carpet, blue walls and blue curtains. The chair will pick up a blue tint from light reflecting off of other parts of the room. Also notice that the shadowed areas of your surroundings are not totally dark even if no light source shines directly on the surface. Diffuse light reflecting off of other objects fills in the shadows. Typically ray-tracing uses a trick called ambient light to simulate such effects but it is not very accurate.

Radiosity calculations are only made when a radiosity block is used inside the global_settings block.

The following sections describes how radiosity works, how to control it with various global settings and tips on trading quality vs. speed.

7.5.2 How Radiosity Works

The problem of ray-tracing is to figure out what the light level is at each point that you can see in a scene. Traditionally, in ray tracing, this is broken into the sum of these components:

Diffuse the effect that makes the side of things facing the light brighter;

Specular the effect that makes shiny things have dings or sparkles on them;

Reflection the effect that mirrors give; and

Ambient the general all-over light level that any scene has, which keeps things in shadow from being pure black.

POV-Ray's radiosity system, based on a method by Greg Ward, provides a way to replace the last term - the constant ambient light value - with a light level which is based on what surfaces are nearby and how bright in turn they are.

The first thing you might notice about this definition is that it is circular: the brightness and color of everything is dependent on everything else and vice versa. This is true in real life but in the world of ray-tracing, we can make an approximation. The approximation that is used is: the objects you are looking at have their ambient values calculated for you by checking the other objects nearby. When those objects are checked during this process, however, their diffuse term is used. The brightness of radiosity in POV-Ray is based on two things:

1. the amount of light "gathered"
2. the 'diffuse' property of the surface finish

An object can have both radiosity and an ambient term. However, it is suggested that if you use radiosity in a scene, you either set ambient_light to 0 in global_settings, or use ambient 0 in each object's finish. This lighting model is much more realistic, and POV-Ray will not try to adjust the overall brightness of the radiosity to match the ambient level specified by the user.

How does POV-Ray calculate the ambient term for each point? By sending out more rays, in many different directions, and averaging the results. A typical point might use 200 or more rays to calculate its ambient light level correctly.

Now this sounds like it would make the ray-tracer 200 times slower. This is true, except that the software takes advantage of the fact that ambient light levels change quite slowly (remember, shadows are calculated separately, so sharp shadow edges are not a problem). Therefore, these extra rays are sent out only

once in a while (about 1 time in 50), then these calculated values are saved and reused for nearby pixels in the image when possible.

This process of saving and reusing values is what causes the need for a variety of tuning parameters, so you can get the scene to look just the way you want.

7.5.3 Adjusting Radiosity

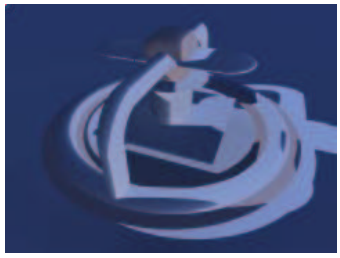
As described earlier, radiosity is turned on by using the radiosity { } block in global_setting. Radiosity has many parameters that are specified as follows:

```
  global_settings { radiosity { [RADIOSITY_ITEMS...] } }

RADIOSITY_ITEMS:
  adc_bailout Float | always_sample Bool | brightness Float |
  count Integer | error_bound Float | gray_threshold Float |
  load_file Filename | low_error_factor Float |
  max_sample Float |
  media Bool | minimum_reuse Float | nearest_count Integer |
  normal Bool | pretrace_end Float | pretrace_start Float |
  recursion_limit Integer | save_file Filename
```

7.5.4 Radiosity with conventional lighting


No radiosity


With radiosity


Difference between them

From [pov:4.1.2] :
You can see that radiosity much affects the shadowed parts when applied combined with conventional lighting.

Changing brightness changes the intensity of radiosity effects. brightness 0 would be the same as without radiosity. brightness 1 should work correctly in most

cases, if effects are too strong you can reduce this. Larger values lead to quite strange results in most cases

Changing the recursion_limit value leads to the following results, second line are difference to default (recursion_limit 3): recursion_limit 1

You can see that higher values than the default of 3 do not lead to much better results in such a quite simple scene. In most cases values of 1 or 2 are sufficient.

The error_bound value mainly affects the structures of the shadows. Values larger than the default of 1.8 do not have much effects, they make the shadows even flatter. Extremely low values can lead to very good results, but the rendering time can become very long. For the following samples recursion_limit 1 is used. error_bound 0.01

Somewhat related to error_bound is low_error_factor. It reduces error_bound during the last pretrace step. Changing this can be useful to eliminate artefacts. low_error_factor 0.01

The next samples use recursion_limit 1 and error_bound 0.2. These 3 pictures illustrate the effect of count. It is a general quality and accuracy parameter leading to higher quality and slower rendering at higher values. count 2

Another parameter that affects quality is `nearest_count`. You can use values from 1 to 20, default is 5: `nearest_count 1`

Again higher values lead to less artefacts and smoother appearance but slower rendering.

minimum_reuse influences whether previous radiosity samples are reused during calculation. It also affects quality and smoothness. minimum_reuse 0.2

Another important value is pretrace_end. It specifies how many pretrace steps are calculated and thereby strongly influences the speed. Usually lower values lead to better quality, but it's important to keep this in good relation to error_bound. pretrace_end 0.2

Strongly related to pretrace_end is always_sample. Normally even in the final trace additional radiosity samples are taken. You can avoid this by adding always_sample off. That's especially useful if you load previously calculated radiosity data with load_file. always_sample on

The effect of max_sample is similar to brightness. It does not reduce the radiosity effect in general but weakens samples with brightness above the specified value. max_sample 0.5

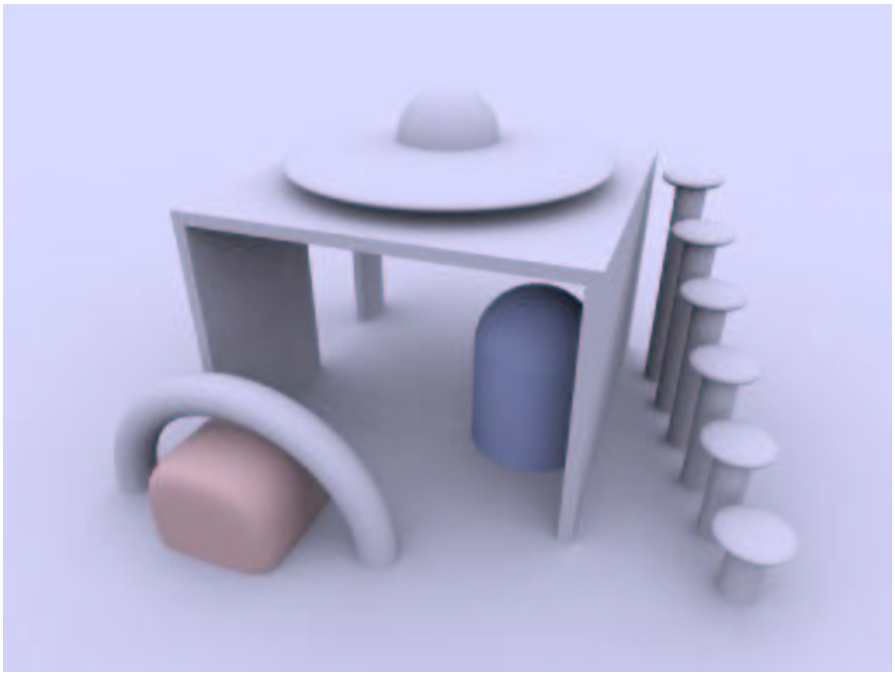
You can strongly affect things with the objects' finishes. In fact that is the most important thing about radiosity. Normal objects should have ambient finish 0

which is not default in POV-Ray and therefore needs to be specified. Objects with ambient ¿ 0 actually emit light.

Finally you can vary the sky in outdoor radiosity scenes. In all these examples it is implemented with a sphere object. finish { ambient 1 diffuse 0 } was used until now. The following pictures show some variations:

7.5.5 Radiosity without conventional lighting

From [pov:4.1.3] :
You can also leave out all light sources and have pure radiosity lighting. The situation then is similar to a cloudy day outside, when the light comes from no specific direction but from the whole sky.



Radiosity without light sources

See also

- [pov:4.1.4] Normals and Radiosity
- [pov:4.1.5] Performance considerations

Bibliography

[BGP83] J.E. Bresenham, D.G. Grice, and S.C. Pi. Bi-directional display of circular arcs. *US.Patent 4 371 933*, 1983. 36

[Bre65] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965. 32

[Bre77] J.E. Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2):100–106, 1977. 36

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley Publ. Comp., Reading, Massachusetts, second edition, 1990. 30, 31, 33, 35, 36, 37, 38, 39, 40, 67, 84, 131

[Kri02] Andreas Kriegl. Mathematik 1 für Informatik LA. Lecture Notes, Vienna, Februar 2002. 78

[Pit67] M.L.V. Pitteway. Algorithm for drawing ellipses or hyperbolae with a digital plotter. *Computer J.*, 10(3):282–289, 1967. 32

[PK87] Roy A. Plastock and Gordon Kalley. *Computergrafik*. Schaum Überblicke. McGraw-Hill Book Comp. GmbH, Hamburg, 1987. 30, 35, 37, 39, 67, 84

[VA84] J.R. Van Aken. An efficient ellipse-drawing algorithm. *CG & A*, 4(9):24–35, 1984. 32

Index