UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Computer Graphics

Introduction, Colour Perception and Colour Models

FH-Prof. Dr. techn. Dipl.-Inf. (FH) Christoph Anthes, MSc

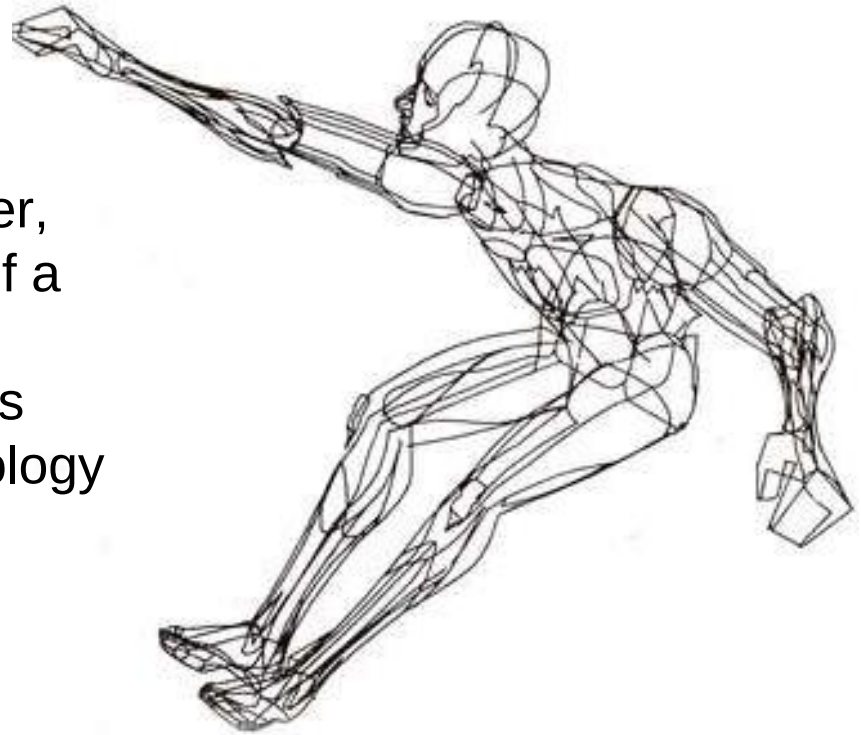
Professor of Augmented and Virtual Reality

Overview

- Introduction
  - General Concepts
  - History
- Anatomy of the Eye
  - Colour Perception
- Colour Models
  - CIE
  - RGB/RGBA
  - CYMK
  - HSL/HSV
  - Conversion Between Colour Models

Introduction

- Definition
  - Computer graphics are graphics created using computers and, more generally, the representation and manipulation of pictorial data by a computer
  - “Perhaps the best way to define computer graphics is to find out what it is not. It is not a machine. It is not a computer, nor a group of computer programs. It is not the know-how of a graphic designer, programmer, a writer, a motion picture specialist, or a reproduction specialist. Computer graphics is all these – a consciously managed and documented technology directed toward communicating information accurately and descriptively.”
    - *William Fetter 1966*



http://dada.compart-bremen.de/item/artwork/240

Relevance of Computer Graphics

- Foundation for graphical digital media and print media
  - 2D/3D interfaces, web pages, apps, newspapers, magazines, posters, advertisements, info graphics

https://hive.fh-hagenberg.at/

- Foundation for special effects and computer generated movies
  - SFX in blockbusters, animated movies from Pixar and others, but also simple TV like news, game shows, sports

https://amp.thisisinsider.com/images/5b6a78efbda1c71a008b4623-750-375.jpg

- Foundation for computer games
  - PC, consoles, mobile games, different rendering styles real-time aspects important

https://icdn6.digitaltrends.com/image/red-dead-redemption-2-review-29850-1920x1080.jpg

- Foundation for virtual and augmented reality
  - Here the additional aspect of stereoscopy plays an important role (covered in AR and VR lecture - optional course)

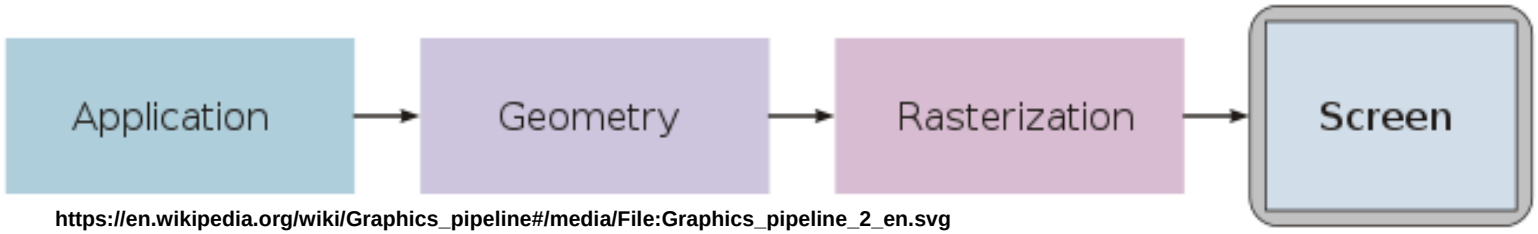
https://www.wxyz.com/entertainment/tesla-suit-unveiled-at-ces-2019-takes-virtual-reality-to-new-heights

2D representation of 3D content

- The term computer graphics (CG) is considered the inverse of computer vision (CV) – CG generates images, CV analyses images
- Typically a display is two-dimensional
- Exceptions
  - For 3D representation two eyes and separate images per eye are required (stereo perception)
  - Volumetric displays, autostereoscopic displays, Virtual Reality and Augmented Reality stereo displays exists – stereoscopic displays (more details in the AR and VR lecture available)
- We will focus in this lecture on two-dimensional display of graphics
- The underlying graphics model discussed in the lecture is in most cases three-dimensional
- Focus in this course 3D graphics, or in detail, the generation of 2D images based on data representing three-dimensional geometrical structures

Graphics Pipeline

- Rendering pipeline or graphics pipeline is a model in computer graphics
- It describes the processing steps from mathematical model to the displayed image

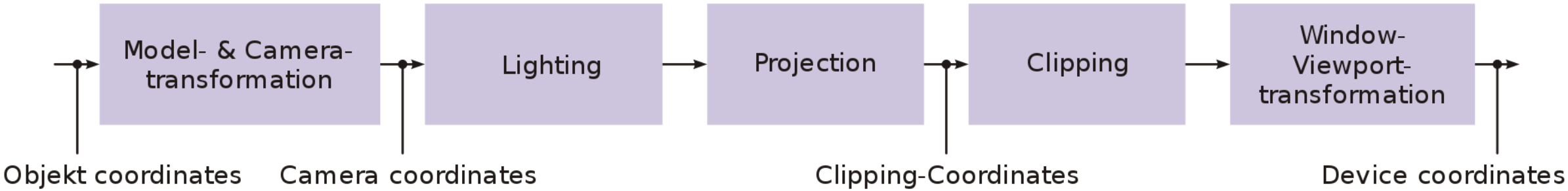


https://en.wikipedia.org/wiki/Graphics_pipeline#/media/File:Graphics_pipeline_2_en.svg

- Extremely simplified model (shown above)
  - **Application** – provides the mathematical model and the manipulations applied on it, typically performed on CPU can contain parts like interaction, animation, physics simulation
  - **Geometry** – main focus of the lecture, processing of the geometry, consists of lighting, transformations, projection, clipping etc.
  - **Rasterisation** – process of generating a 2D image based on the model defined in the geometry stage
  - **Screen** is the final display medium (2D display in our case)

Graphics Pipeline

- A more detailed look at the geometry stage

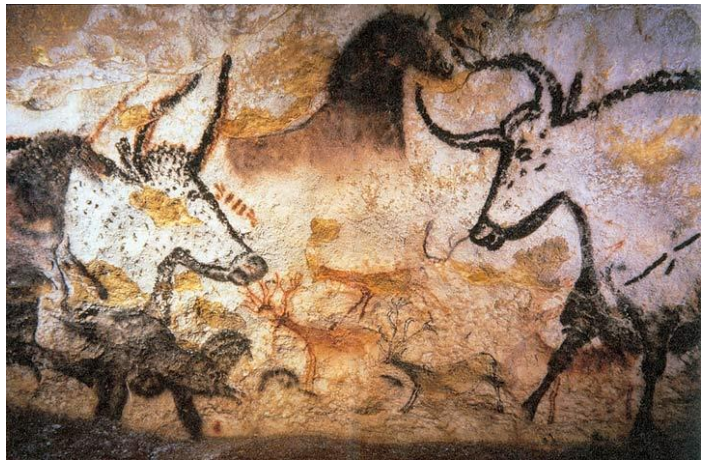


https://en.wikipedia.org/wiki/Graphics_pipeline#/media/File:Geometry_pipeline_en.svg

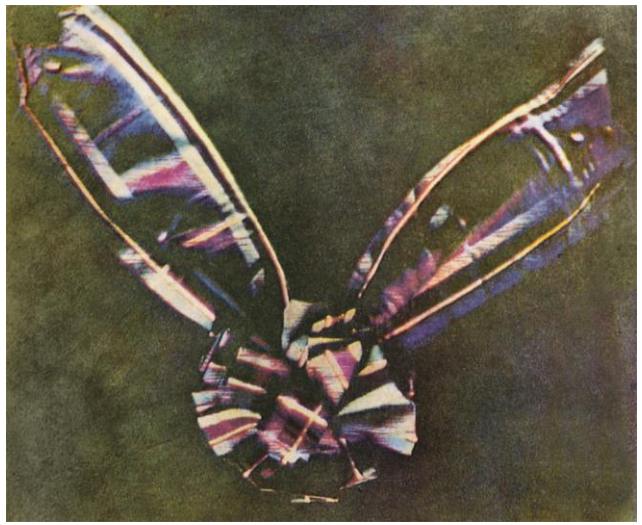
- Model & camera transformation – placement of objects in world coordinate system, camera coordinate system
- Lighting and shading of objects determination of colour
- Projection on 2D plane
- Removal of elements out of sight
- Transformation into device coordinates

History

- Early History
  - 30.000 BC - Early cave paintings in Lascaux (France)
- Renaissance
  - 1428 - Perspective drawing by Filippo Brunelleschi
- Industrial Revolution
  - 3 colour process for printing
  - 1861 - Early colour photography by Maxwell uses three filters one for each of the trichromatic colours



https://upload.wikimedia.org/wikipedia/commons/1/1e/Lascaux_painting.jpg

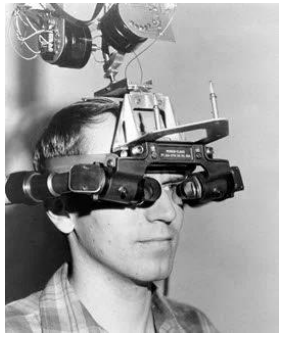


http://www.clerkmaxwellfoundation.org/html/first_colour_photographic_image.html

https://en.wikipedia.org/wiki/CMYK_color_model#/media/File:NIEdot367.jpg

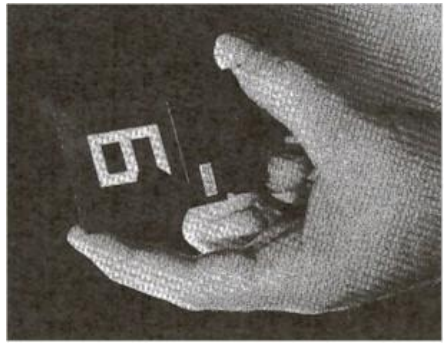
History

- Displays and Interaction
  - 1926 first TV by Kenjiro Takayanagi
    - Cathode Ray Tube (CRT) technology already developed by Braun in 1897
  - 1963 Ivan Sutherland's Sketchpad
    - Direct interconnection between display and user
  - 1964 Donald L. Bitzer
    - The first plasma display panel (PDP) later on integrated into PLATO (Programmed Logic for Automatic Teaching Operations)
  - 1968 Ivan Sutherland's Sword of Damocles
    - First stereoscopic Virtual Reality Head-Mounted Display (actually it is an Augmented Reality (AR) Ceiling Mounted Display)



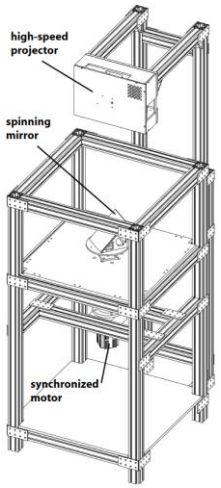
History

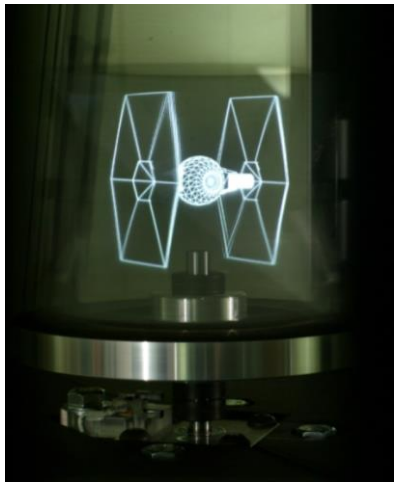
- Displays and Interaction
  - 1968 First LCD introduced by Radio Corporation of America (RCA), the origins date back to Otto Lehmann 1904
  - 1997 Plasma display by Pioneer
  - OLED (organic light emitting diode) technology originates from the 1950s (A. Bernanose) first commercial displays were produced 2009

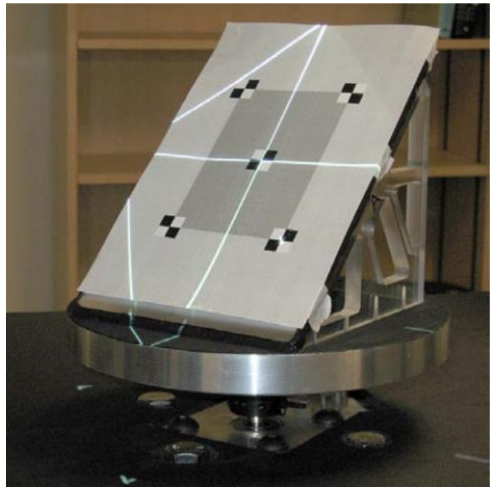


https://ethw.org/File:Liquid_Crystal_Display.jpg

  - 1996 Light Field Rendering by Mark Levoy and Pat Hanrahan
  - 2007 Light field Display by Jones et al. allows three-dimensional display of light fields



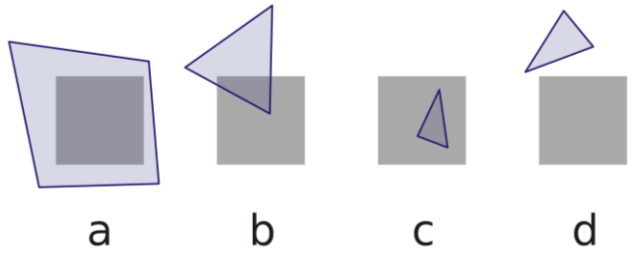




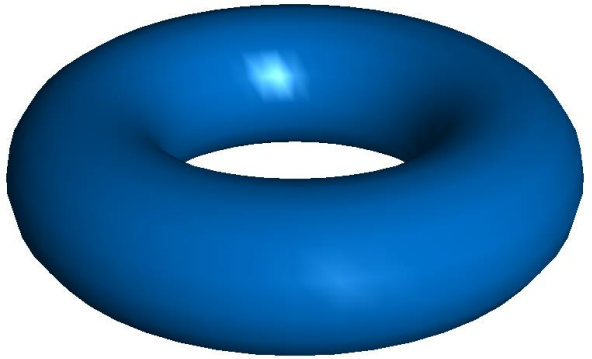
Jones, A.; McDowall, I.; Yamada, H.; Bolas, M. & Debevec, P., Rendering for an interactive 360textdegree light field display *ACM Transactions on Graphics, Association for Computing Machinery (ACM),* 2007, *26*, 40

History

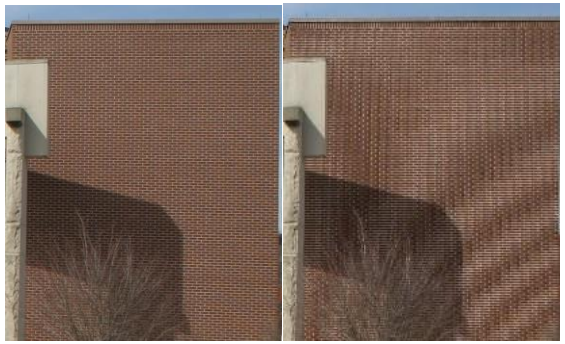
- Algorithms – vector graphics
  - Roberts (1963), Appel (1967) - hidden-line algorithms
  - Warnock (1969), Watkins (1970) - hidden-surface algorithms
  - Sutherland (1974) - visibility = sorting
- 1970s - raster graphics
  - Gouraud (1971) - diffuse lighting
  - Phong (1974) - specular lighting
  - Blinn (1974) - curved surfaces, texture bump mapping
  - Catmull (1974) - Z-buffer hidden-surface algorithm
  - Crow (1977) - anti-aliasing
- 1973 first SIGGRAPH (the place to be)
  - Biggest and most cited Computer Graphics Conference
  - First event organised by Jon Meads and Bob Schiffman



https://de.wikipedia.org/wiki/Warnock-Algorithmus#/media/File:Warnock1.svg



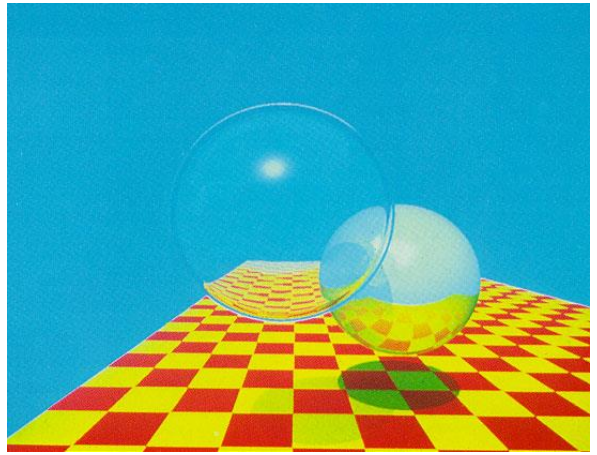
https://en.wikipedia.org/wiki/Gouraud_shading#/media/File:Gouraudshading00.png



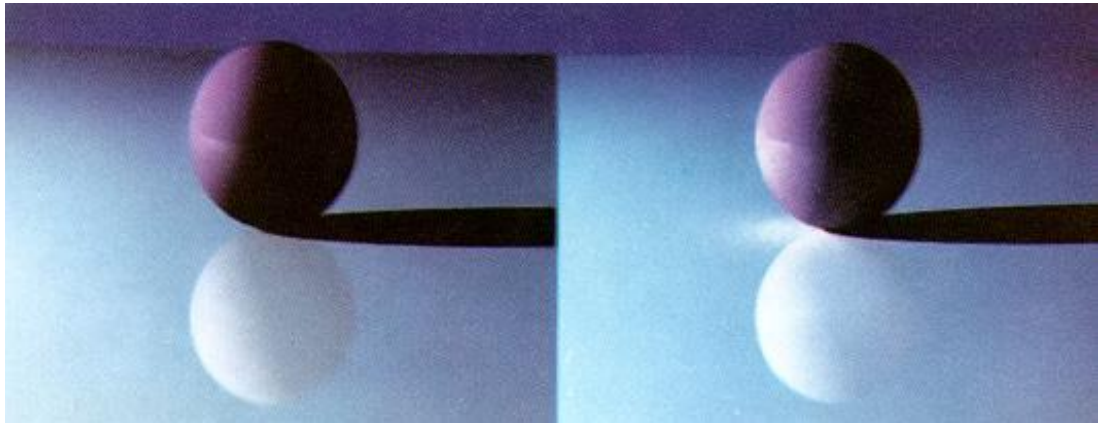
https://en.wikipedia.org/wiki/Aliasing#/media/File:Moire_pattern_of_bricks.jpg

History

- Global Illumination
  - Appel (1963) – original idea of the ray tracing algorithm to create realistic graphics
  - Whitted (1980) – recursive ray tracing
  - Goral, Torrance et al. (1984), Cohen (1985) – radiosity algorithm based on energy distribution inside cells instead of following rays
  - Kajiya (1986) – the rendering equation
- Photorealism
  - Cook (1984) – shade trees
  - Perlin (1985) – shading languages
  - Hanrahan and Lawson (1990) – RenderMan



Whitted, T. An improved illumination model for shaded display Communications of the ACM, Association for Computing Machinery (ACM), 1980, 23, 343-349



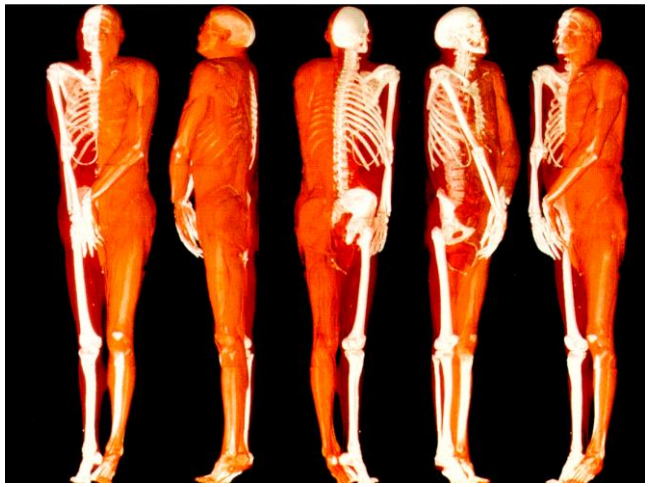
Kajiya, J. T. The rendering equation ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1986, 20, 143-150

Cook, R. L. Shade trees Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH '84, ACM Press, 1984

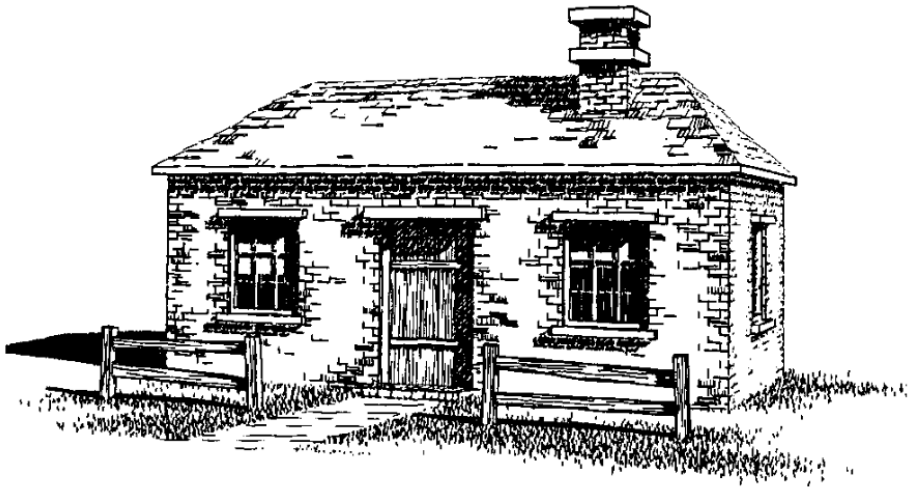
History

- Early 1990s - non-photorealistic rendering
  - Drebin et al. (1988), Levoy (1988) – volume rendering
  - Haeberli (1990) – impressionistic paint programs
  - Salesin et al. (1994-) – automatic pen-and-ink illustration
  - Meier (1996) – painterly rendering

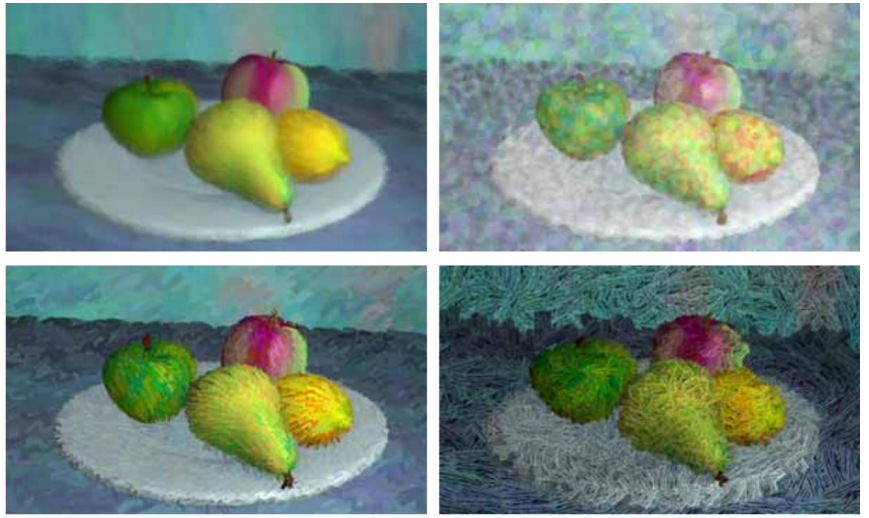
Haeberli, P. Paint by numbers: abstract image representations Proceedings of the 17th annual conference on Computer graphics and interactive techniques - SIGGRAPH '90, ACM Press, 1990



Drebin, R. A.; Carpenter, L. & Hanrahan, P. Volume rendering Proceedings of the 15th annual conference on Computer graphics and interactive techniques - SIGGRAPH '88, ACM Press, 1988



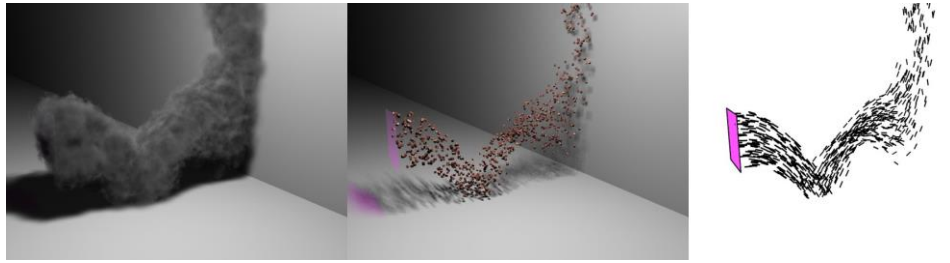
Winkenbach, G. & Salesin, D. H. Computer-generated pen-and-ink illustration Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94, ACM Press, 1994



Meier, B. J. Painterly rendering for animation Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96, ACM Press, 1996

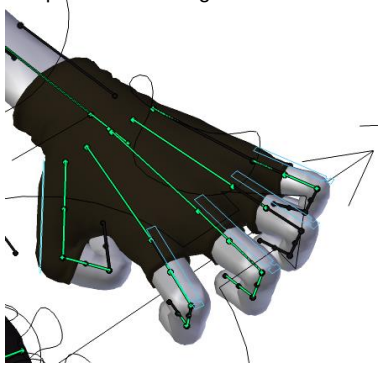
History

- Animation
  - 1983 – Particle Systems (William T. Reeves)
    - Used to animate amorph surfaces like fire, smoke, dust, water
  - 1987 – Swarming and Flocking (Craig W. Reynolds)
    - Animation of group of characters (steering behaviours), initially flocks of birds, schools of fish (boids)
    - Later on adapted and used for crowd animation as well
  - 1988 – Skeletal animation (Nadja Magnenat-Thalmann et al.)
    - Used to animate characters or parts of characters by providing a skeleton which is used to animate the surface of these characters
    - Common procedure used in current games and SFX
    - Also known as mesh skinning



https://de.wikipedia.org/wiki/Partikelsystem#/media/File:Particles.jpg

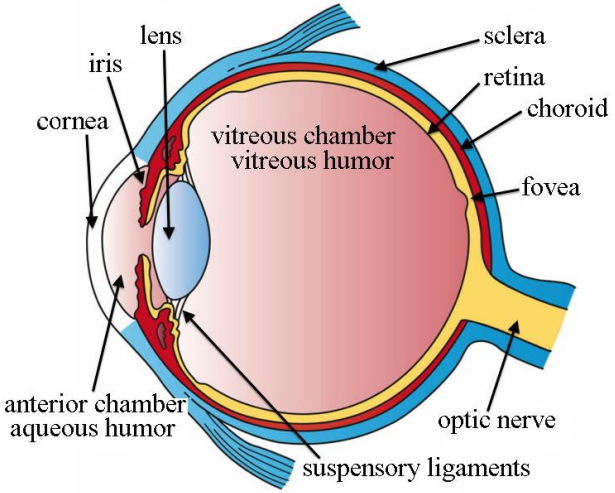
https://www.indiewire.com/wp-content/uploads/2016/06/game-of-thrones-609.jpg?w=780



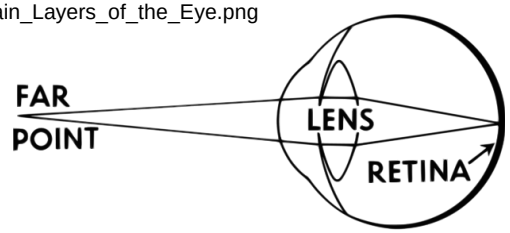
https://en.wikipedia.org/wiki/Skeletal_animation#/media/File:Sintel-hand_(cropped).png

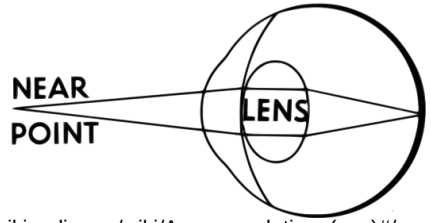
Anatomy of the Eye

- *Retina* contains rods and cones (brightness and colour)
- *Fovea centralis* is area where the images are perceived sharpest during daylight
- Light on *optic nerve* is not perceived (blind spot)
- *Choroid* provides support of oxygen and nutrition
- *Sclera* acts as protective layer, becomes transparent *cornea* in the front area before *iris* and *lens*
- Accommodation is the process of changing the *lens* with the help of the *suspensory ligaments* in order to produce a sharp image on *retina*, depending on the distance to the object
  - The closest near point of a healthy eye is at around 25cm



https://en.wikipedia.org/wiki/Human_eye#/media/File:Three_Main_Layers_of_the_Eye.png

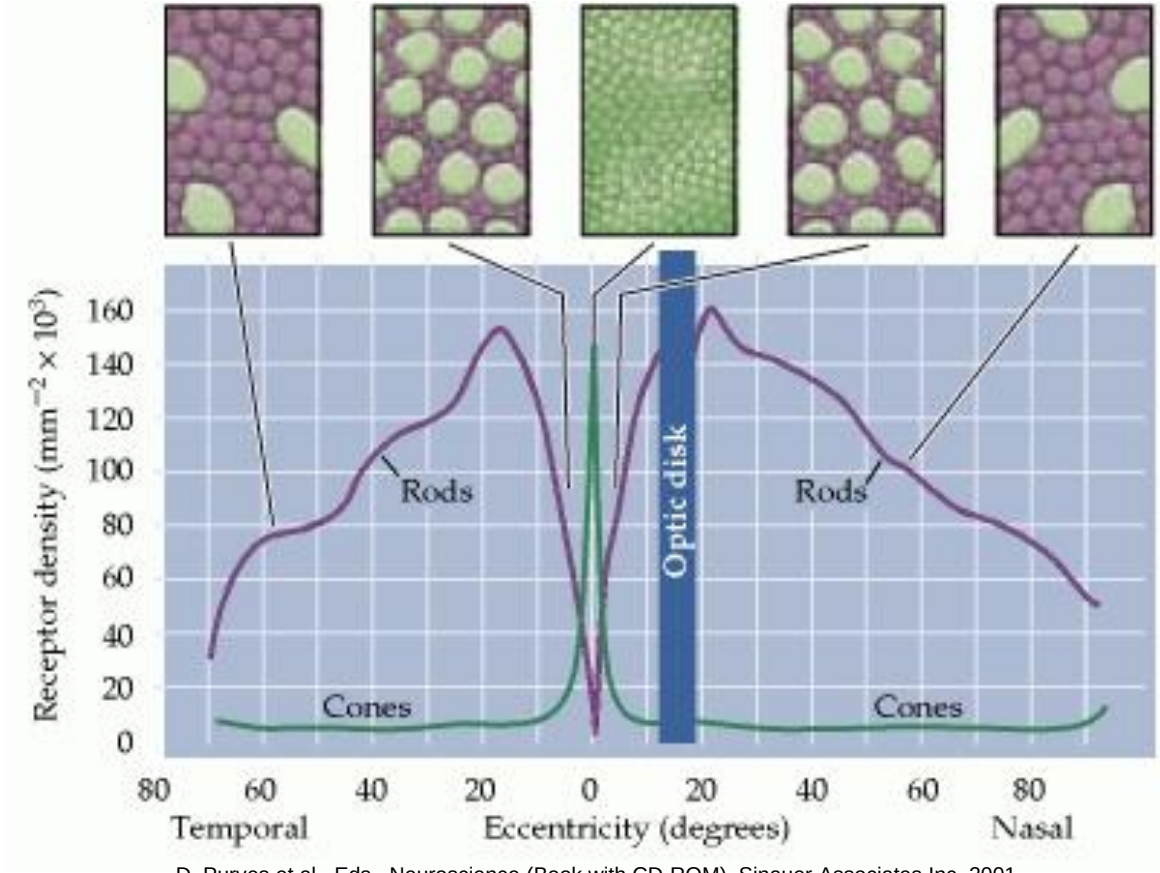




https://en.wikipedia.org/wiki/Accommodation_(eye)#/media/File:Accommodation_(PSF).svg

Anatomy of the Eye

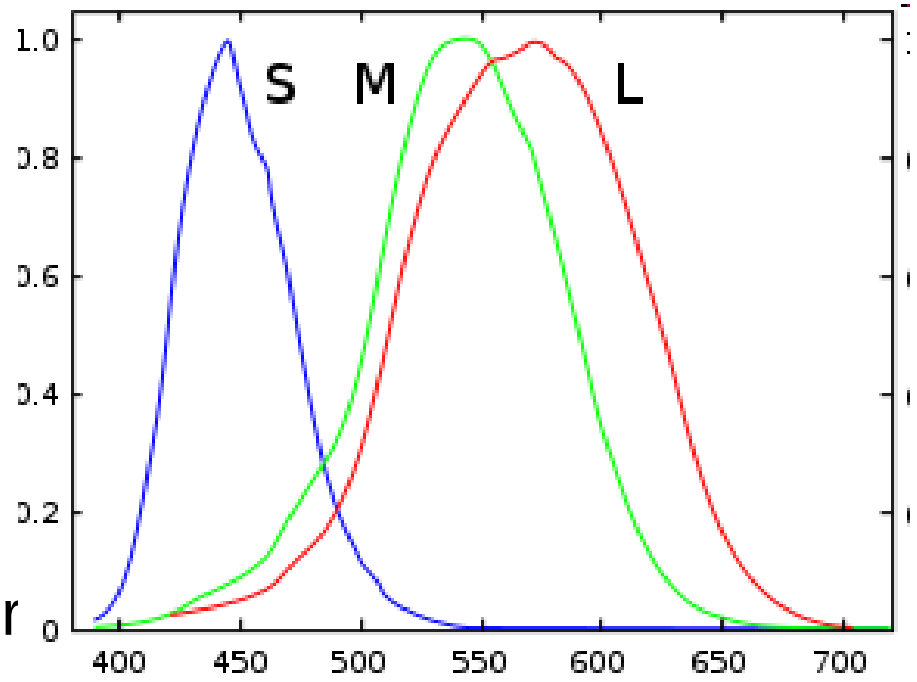
- Adaption is the process of adjusting the eye based on the light intensities
  - During adaption the pupil size can be changed (smaller in brightness, larger in darkness), sensitivity of rods and cones is also affected
  - A full adaptation process from very bright to dark can take between 20-30 minutes, from dark to bright 20-60 seconds
- Rods are used for brightness perception
- Cones are used for colour perception
  - Red – long wavelength
  - Green – medium wavelength
  - Blue – short wavelength
- Both are distributed in different ways around the fovea



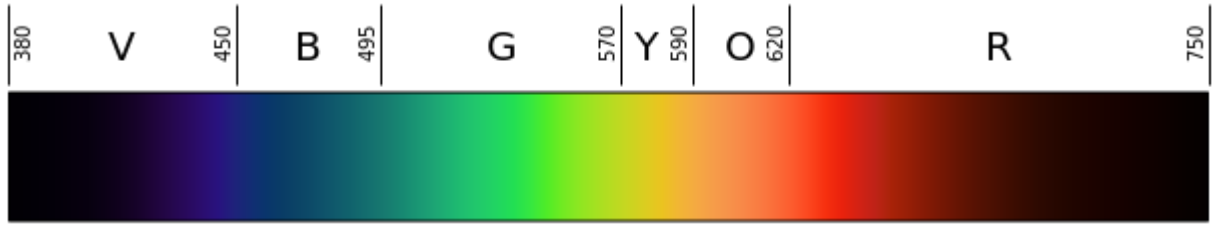
D. Purves et al., Eds., Neuroscience (Book with CD-ROM). Sinauer Associates Inc, 2001.

Visual Perception

- Colour is the way how we perceive waves of light
- Perception of an overlap of electromagnetic waves of different wave lengths
- Focus on colour perception not visual perception in general
- Visible electromagnetic spectrum
  - Ranges from 380 nm (violet) to 740 nm (red)
- Shorter wavelengths (ultraviolet, x-ray) and longer wavelengths (infrared, radio) are not perceived by the human eye
- Not only wavelength but also intensity is relevant for perception



https://en.wikipedia.org/wiki/Cone_cell#/media/File:Cones_SMJ2_E.svg



https://en.wikipedia.org/wiki/Visible_spectrum#/media/File:Linear_visible_spectrum.svg

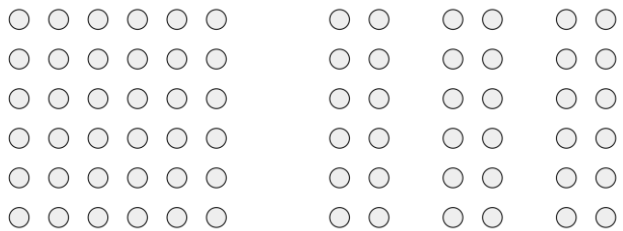
Visual Perception

- Human Visual System (HSV) is not well understood, knowledge about the eye is comparably easy, brain is hard to understand
- Variety of models, theories, measurement techniques exist which draw background from domains of physics, physiology, perceptual psychology, art and graphics design
- The perceived colour of an object depends on more than the object itself, it is dependant on the surrounding light sources, reflections from the surrounding area, the human visual system
- Light can be **reflected** from objects (e.g. wall, desk) or it can be **transmitted** (e.g. glass, water)
  - If a surface reflects only pure blue and is illuminated by pure red it will appear black
  - If a transmissive surface only transmits pure green light and a pure blue light source is viewed through the surface it will appear black

Visual Perception

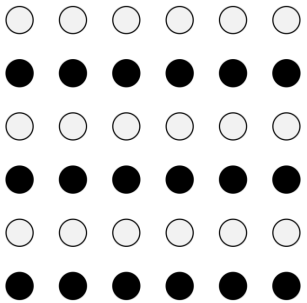
- Different levels of processing in visual perception
  - Early vison is low-level processing, perceiving sharp contrasts in brightness, small changes in orientation and colour, spatial frequencies and edge detection
  - Mid-level to high-level processing provides shape and object recognition, motion detection, attention handling and eye control
  - High-level processing
- Gestalt theory provides a set of laws for high-level processing

Proximity



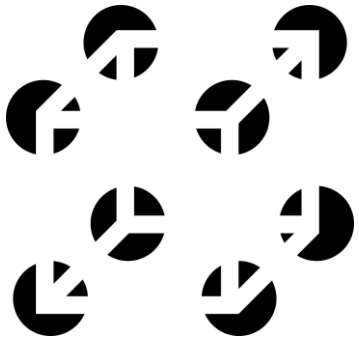
https://de.wikipedia.org/wiki/Gestaltpsychologie#/media/File:Gestalt_proximity.svg

Similarity



https://de.wikipedia.org/wiki/Gestaltpsychologie#/media/File:Gestalt_proximity.svg

Closure



https://de.wikipedia.org/wiki/Gestaltpsychologie#/media/File:Nocube.svg

Continuation

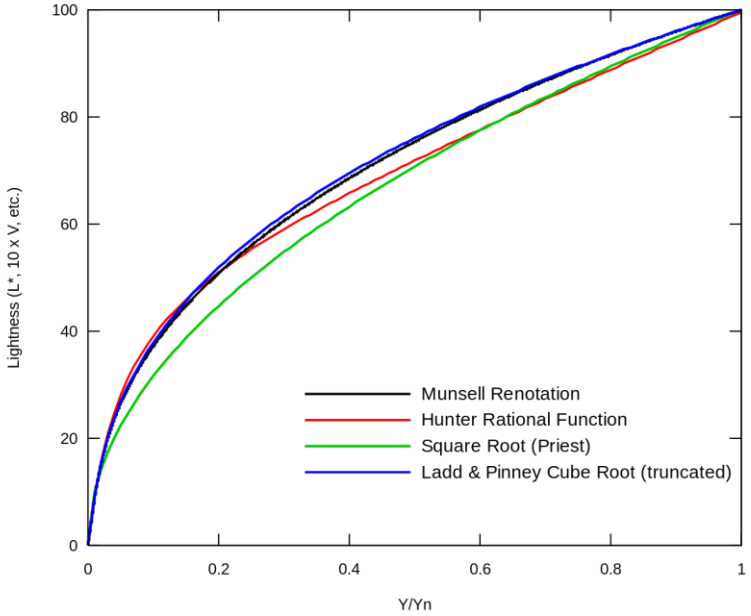
https://de.wikipedia.org/wiki/Gestaltpsychologie#/media/File:Gesetz_der_guten_Fortsetzung.png

Symmetry

https://en.wikipedia.org/wiki/Gestalt_psychology#/media/File:Law_of_Symmetry.jpg

Visual Perception

- Non-linear correlation between intensity and brightness
  - The HVS is non-linear and roughly based on ratios
  - In general humans are more sensitive to differences at lower levels of illumination
  - Difference in intensity/luminance I e.g. I=0.10 and 0.11 is perceived the same as between I=0.50 and 0.55



- In relation to the emitted energy roughly logarithmic relation to perceived light
  - E.g. light bulbs 25W, 50W, 100W seem linear to the human eye

Colours

- Colour Palettes
  1. Perceivable colours
  2. Displayable colours on a monitor or other display (typically subset of 1.)
  3. Calculable colours by a compute system (typically subset of 1. but more available colours than 2.)
- Colour Depth
  - Describes the amount of colours which can be displayed or stored in an image file
  - Colour values are typically encoded in integer or byte values
  - Colour depth is always given in powers of two
  - When using red, green, blue (RGB) , 8 bits per colour channel should be sufficient resulting in $2^{24} = 16.777.216$ colours (true colour)
  - Problem: we still have discretisation of colour

Colour Models

- Saturation
  - Colorfulness of an area judged in proportion to its brightness
  - Pure red, blue … have a high amount of saturation
  - Pastel colours have a low amount of saturation
- Intensity, lightness or luminosity
  - Light intensity which is actually emitted
- Brightness or lightness
  - Light intensity which is perceived
- Hue
  - Dominant wavelength of a colour (e.g. blue, red)

Black-White and Greyscale Models

- Binary encoding of an image uses 1 bit per pixel (point on the screen), either black or white (BW)
- Origin of the term bitmap image
- Variety of processing techniques from image processing to convert from colour space into bitmap image (e.g. dithering)
- Memory consumption of an 1024x768 BW image

$$\frac{1024 \cdot 768}{8 \cdot 1024} = 96 KBytes$$

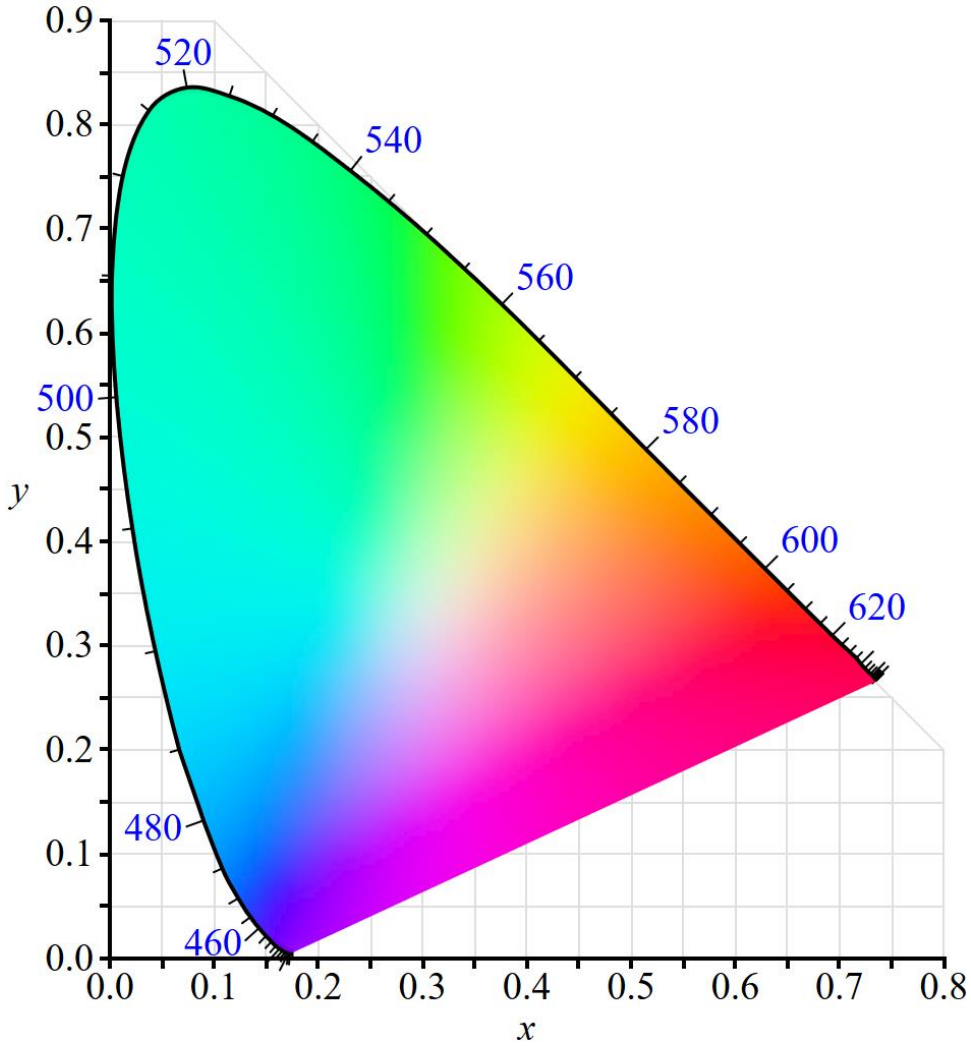
https://upload.wikimedia.org/wikipedia/en/3/33/Neighborhood_watch_bw.png

- Greyscale image makes use of a range of grey values between black and white
- Memory consumption of an 1024x768 8bit greyscale image (256 shades of grey)

$$\frac{1024 \cdot 768 \cdot 8}{8 \cdot 1024} = 768 KBytes$$

Commision Internationale de L'Eclairange (CIE)

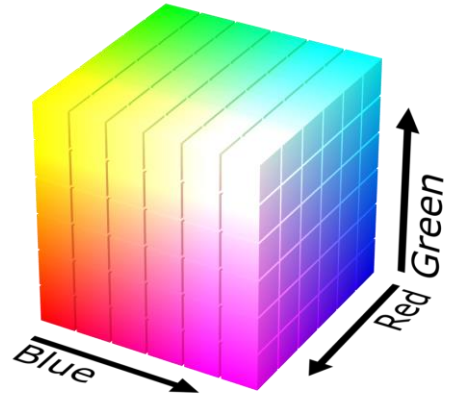
- Physically based colour model
- Defined by CIE in 1931 http://www.cie.co.at/
- Theoretical colour values x,y and z are projected in an x-y plane
- Only hue and saturation are considered, intensity is omitted
- Unfortunately not really applicable to current display technology, thus less relevant in computer graphics
- Colorimeters and spectroradiometers use CIE
- Relevant in industry (paint, lighting, physics, chemistry)



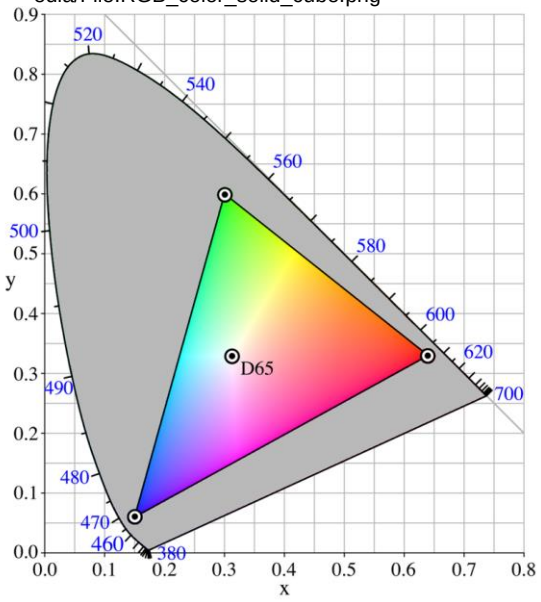
https://en.wikipedia.org/wiki/CIE_1931_color_space#/media/File:CIE1931xy_blank.svg

Red – Green – Blue (RGB)

- Colour model closely related to display technology (CRT, LCD, plasma, OLED, quantum dots, etc.), thus the most relevant colour model for computer graphics
- Additive colour model
- Additional colour channel describing the transparency (RGBA)
- Description can be mapped on a cube, complementary colours are on the opposite side of the cube
  - Origin (0,0,0) is black
  - (1,1,1) is white
  - Greyscale values lie on the diagonal
- In general easy to understand but not necessarily intuitive to describe colour
- Colour gamut only represents subsection of the CIE model



https://en.wikipedia.org/wiki/RGB_color_model#/media/File:RGB_color_solid_cube.png



https://en.wikipedia.org/wiki/RGB_color_model#/media/File:CIExy1931_sRGB_gamut_D65.png

Red – Green – Blue (RGB)

- Colours represented a triplet of red, green, blue
- Different representations are possible (e.g. arithmetic, percentage)
- Different colour depths are possible, which does have an impact on memory consumption and image transfer
- Memory consumption of 8-bit per colour channel

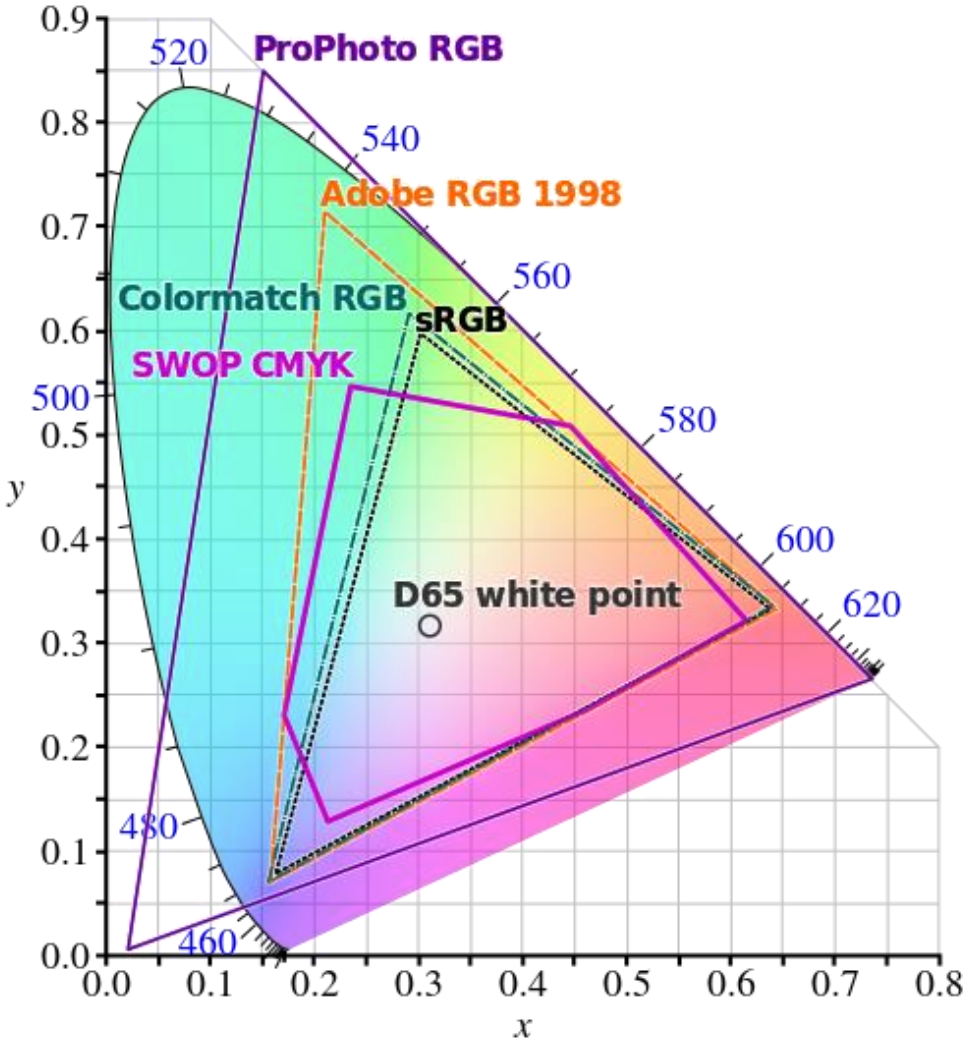
$$\frac{1024 \cdot 768 \cdot 24}{8 \cdot 1024 \cdot 1024} = 2.25 MBytes$$

Notation	RGB triplet
Arithmetic	(1.0, 0.0, 0.0)
Percentage	(100%, 0%, 0%)
Digital 8-bit per channel	(255, 0, 0) or sometimes #FF0000 (hexadecimal)
Digital 12-bit per channel	(4095, 0, 0)
Digital 16-bit per channel	(65535, 0, 0)
Digital 24-bit per channel	(16777215, 0, 0)
Digital 32-bit per channel	(4294967295, 0, 0)

- In general it is important to know that typically images are uncompressed when displayed via a graphics card
- More human understandable colour models are desirable and are available in a variety of modelling tools

Cyan – Magenta – Yellow – Key (CMYK)

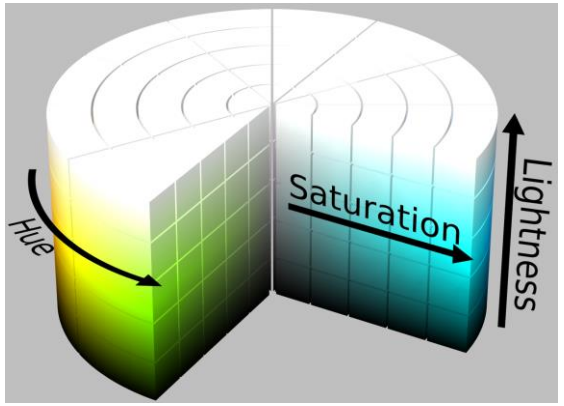
- Closely related to RGB, basically the inverse model plus additional k value
- Main use case in printing (ink or laser printing)
  - For example applying cyan on paper (blue/green) results in reflection of blue and green components no red components are reflected
- Key as a pure black is added in order to save colour
  - It also results in a reduced drying process (ink) since one colour is used instead of three colours
  - Increased speed in printing (drying time reduced)
- Variety of adaptions can cover larger area of the CIE spectrum than traditional RGB space



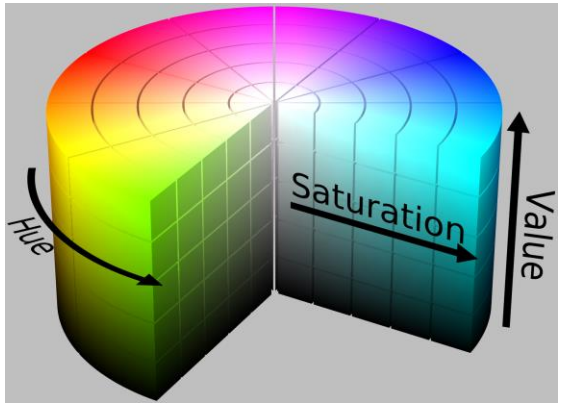
https://en.wikipedia.org/wiki/CMYK_color_model#/media/File:CIE1931xy_gamut_comparison.svg

Hue – Saturation – Lightness/Value (HSL/HSV)

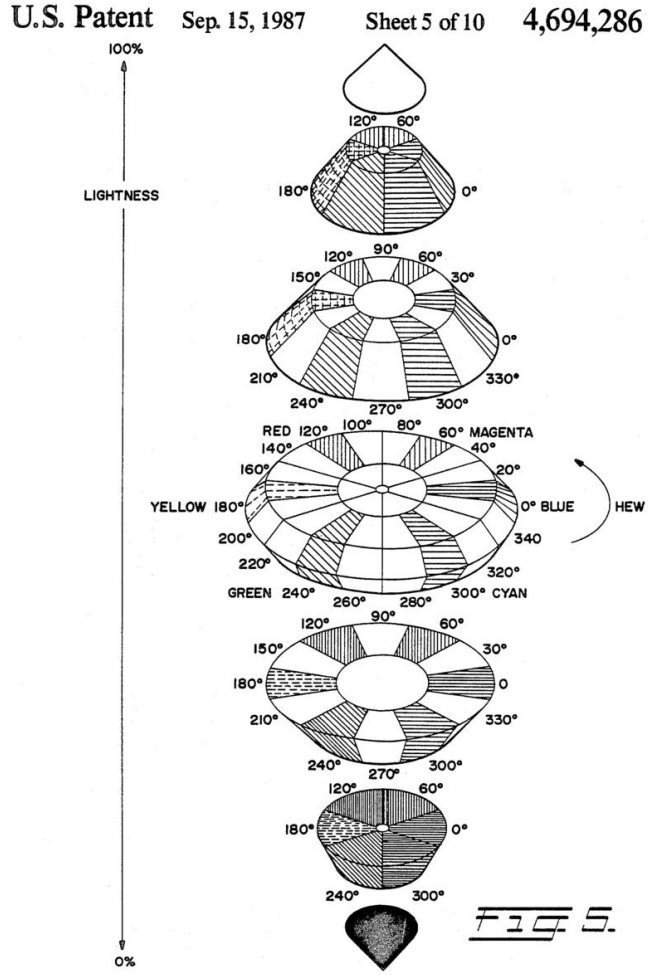
- Alternative representations to adapt RGB model easier for human understanding
- Hue – Saturation – Lightness (HSL)
  - Hue: the actual colour
  - Saturation: the degree how much hue differs from a neutral grey
  - Lightness: the intensity of the colour
- Hue – Saturation – Value (HSV)
  - Hue: the actual colour
  - Saturation: the degree how much hue differs from a neutral grey
  - Value: colour mix from black to white, the displayed Brightness



https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSL_color_solid_cylinder_alpha_lowgamma.png



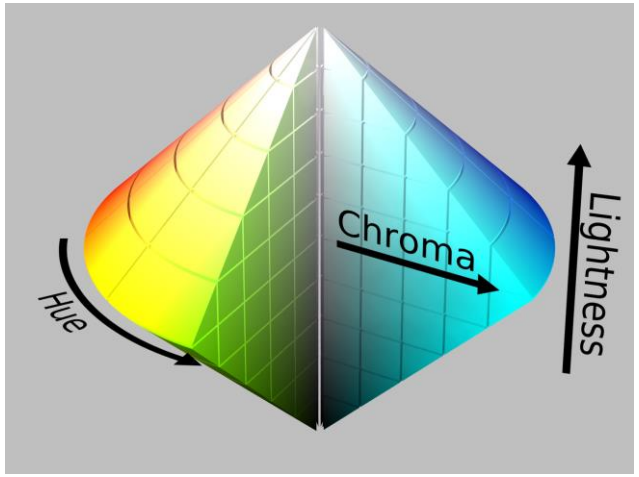
https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSV_color_solid_cylinder_alpha_lowgamma.png



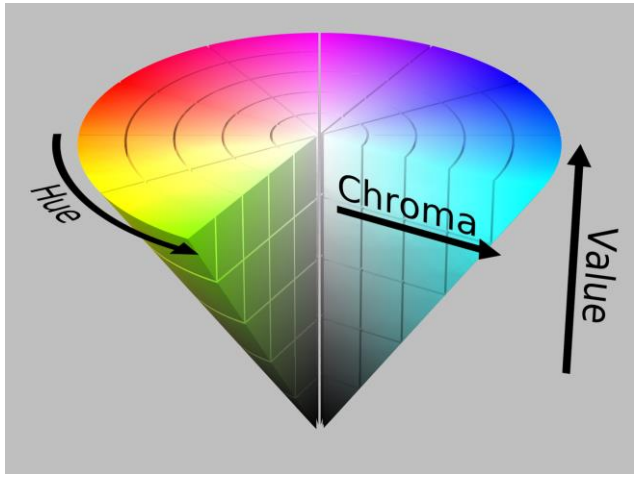
US patent 4694286, Bergstedt, Gar A., "Apparatus and method for modifying displayed color images", published 1987-09-15, assigned to Tektronix, Inc

Hue – Saturation – Lightness/Value (HSL/HSV)

- Problem with cylindrical mapping
  - Redundancy in values with a 0-value of L or V neither hue or saturation are perceivable if cylindrical mapping is used
  - Typical representation uses conical or hex-conical representation
  - S is the distance to the middle axis (L or V)
- HSL uses double cone
  - HSL starts with hue of 0° as blue (a rotation from 120° from HSV)
- HSV uses single cone
  - HSV starts with hue of 0° as red
  - On the surface of the cone saturation is always full opposed to HSL



https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSL_color_solid_dblcone_chroma_gray.png



https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSV_color_solid_cone_chroma_gray.png

Converting Colour Spaces

- RGB to HSI (=HSL)

$$I = \frac{R+G+B}{3}$$

$$S = 1 - \frac{3}{(R+G+B)} \min(R,G,B)$$

$$H = \cos^{-1} \sqrt{\frac{\frac{1}{2}((R-G)+(R-B))}{(R-G)^2+(R-B)(G-B)}}$$

- RGB to CMY

$$[C,M,Y] = [1,1,1] - [R,G,B]$$

- CMY to RGB

$$[R,G,B] = [1,1,1] - [C,M,Y]$$

- Assuming that the individual values range between 0 and 1

Bibliography

- M. H. Weik, *Communications Standard Dictionary*. Van Nostrand Reinhold Company, 1982.
- William Fetter, *Computer Graphics at Boeing.* In: *Print Magazine*, XX:VI, November/Dezember 1966, p. 32
- A. van Dam CSC1 1230 Introduction into Computer Graphics, Brown University 2018
- M. Levoy, History of computer graphics, slideset CS 248 - Introduction to Computer Graphics Autumn quarter, 2004, Slides for September 28 lecture, Stanford
- I. E. Sutherland, “Sketchpad: A man-machine graphical communications system,” MIT Lincoln Laboratories, Technical Report 296, 1963.
- L. G. Roberts, “Machine Perception Of Three-Dimensional Solids,” Lincoln Laboratory, TR 315, MIT, Cambridge, 1963.
- A. Appel, “The notion of quantitative invisibility and the machine rendering of solids,” in *Proceedings of the 1967 22nd national conference on* -, 1967.
- J. Warnock, “A Hidden-Surface Algorithm for Computer Generated Halftone Pictures,” Computer Graphics Department, University of Utah, Salt Lake City, Technical Report TR 4-15, NTIS AD-753 671, 1969.

Bibliography

- G. S. Watkins, “A real time visible surface algorithm,” The University of Utah, 1970.
- E. E. Sutherland, R. F. Sproull, and R. A. Schumacker, “A Characterization of Ten Hidden-Surface Algorithms,” *ACM Computing Surveys*, vol. 6, no. 1, pp. 1–55, Jan. 1974.
- H. Gouraud, “Continuous Shading of Curved Surfaces,” *IEEE Transactions on Computers*, vol. C-20, no. 6, pp. 623–629, Jun. 1971.
- J. F. Blinn, “Simulation of wrinkled surfaces,” *ACM SIGGRAPH Computer Graphics*, vol. 12, no. 3, pp. 286–292, Aug. 1978.
- E. E. Catmull, “A subdivision algorithm for computer display of curved surfaces.,” The University of Utah, 1974.
- F. C. Crow, “The aliasing problem in computer-generated shaded images,” *Communications of the ACM*, vol. 20, no. 11, pp. 799–805, Nov. 1977.
- Whitted, T. An improved illumination model for shaded display Communications of the ACM, Association for Computing Machinery (ACM), 1980, 23, 343-349
- Cook, R. L. Shade trees Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘84, ACM Press, 1984

Bibliography

- Kajiya, J. T. The rendering equation ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1986, 20, 143-150
- W. T. Reeves, “Particle Systems—a Technique for Modeling a Class of Fuzzy Objects,” *ACM Transactions on Graphics*, vol. 2, no. 2, pp. 91–108, Apr. 1983.
- C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, Aug. 1987.
- N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann, “Joint-dependent local deformations for hand animation and object grasping,” *Proceedings of Graphics Interface ’88, Edmonton, Alberta, Canada, 6 - 10 June 1988, 26-33*, 1988.
- Haeberli, P. Paint by numbers: abstract image representations Proceedings of the 17th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘90, ACM Press, 1990
- Drebin, R. A.; Carpenter, L. & Hanrahan, P. Volume rendering Proceedings of the 15th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘88, ACM Press, 1988

Bibliography

- Jones, A.; McDowall, I.; Yamada, H.; Bolas, M. & Debevec, P. Rendering for an interactive 360degree light field display *ACM Transactions on Graphics, Association for Computing Machinery (ACM),* 2007*, 26*, 40
- Winkenbach, G. & Salesin, D. H. Computer-generated pen-and-ink illustration Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘94, ACM Press, 1994
- Meier, B. J. Painterly rendering for animation Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘96, ACM Press, 1996
- D. Purves *et al.*, Eds., *Neuroscience (Book with CD-ROM)*. Sinauer Associates Inc, 2001.
- A. V. Oppenheim, R. W. Schafer, and T. G. Stockham, “Nonlinear filtering of multiplied and convolved signals,” *Proceedings of the IEEE*, vol. 56, no. 8, pp. 1264–1291, 1968.

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Computer Graphics

Modelling and Transformations

FH-Prof. Dr. techn. Dipl.-Inf. (FH) Christoph Anthes, MSc

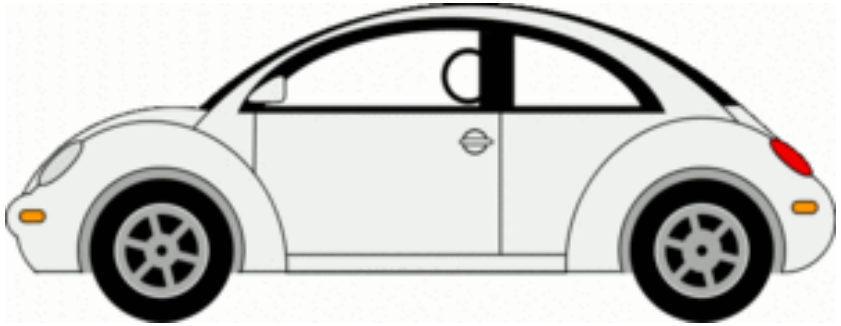
Professor of Augmented and Virtual Reality

Overview

- Geometric Shapes
  - Meshes and Triangles
- Coordinate Systems
- Transformations
  - Multiplying Vectors and Matrices
  - Transformations in 2D Space
  - Concatenating Transformations
  - Transformations in 3D Space
  - Quaternions and Normal Vectors
- Scene Graphs

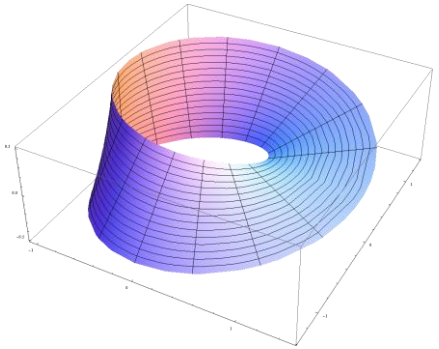
Geometric Shapes

- A **graphics primitive** (aka geometric primitive, prim) is an **atomic** or **irreducible** element to create graphics
- The available primitives depend on the used graphics library
- The most simple primitives a points and lines
- Cubes and polyhedrons are typically no primitives - they can be constructed of lines and points
- Graphics are generated by combining multiple primitives
- Different primitives in 2D and 3D space exist that can be used

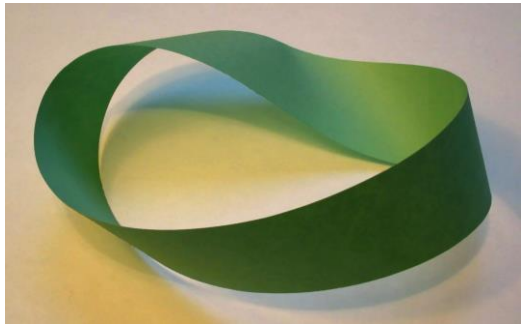


Geometric Shapes

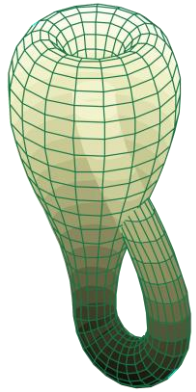
- Idea to render a **three-dimensional solid** by rendering its individual surfaces separated by edges
- 3 attributes of an a three-dimensional solid
  - Closed
  - Orientable
  - No self-intersections
- Non-orientable objects can exist
  - Klein bottle
  - Möbius strip
- Typically in CG we deal with three-dimensional solids



https://de.wikipedia.org/wiki/M%C3%B6biusband#/media/File:M%C3%B6biusband.png



https://de.wikipedia.org/wiki/M%C3%B6biusband#/media/File:M%C3%B6bius_strip.jpg

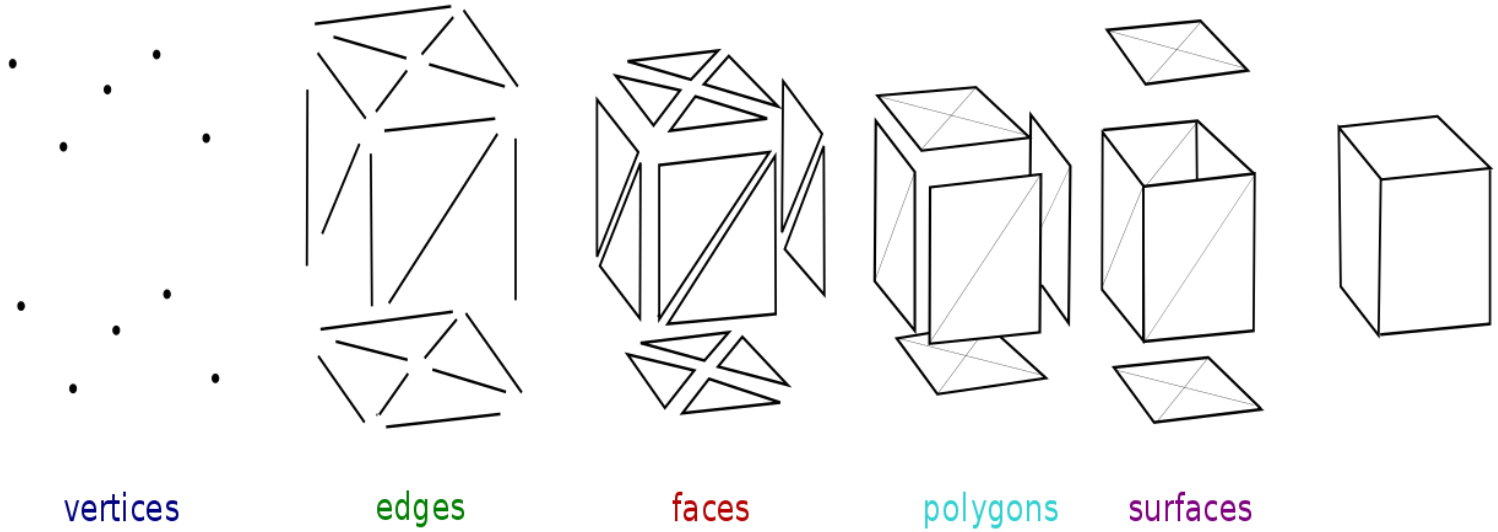


https://de.wikipedia.org/wiki/Kleinsche_Flasche#/media/File:Klein_bottle.svg

https://de.wikipedia.org/wiki/Kleinsche_Flasche#/media/File:Acme_klein_bottle.jpg

Geometric Shapes

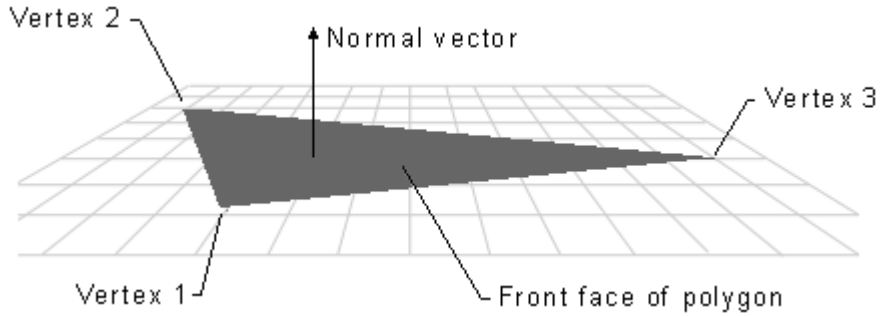
- Typically two-dimensional and three-dimensional surfaces are formed by meshes of triangles (other representations exist e.g. voxels, NURBS)
- Other graphical primitives exist as well (as for example in Constructive Solid Geometry modelling)
- Triangle meshes
  - A triangle (or face) is defined by a set of points (vertex) which is interconnected (edges)
  - Vertices and triangles are characterised by their properties (position, colour, normal vectors, texture coordinates, etc.)

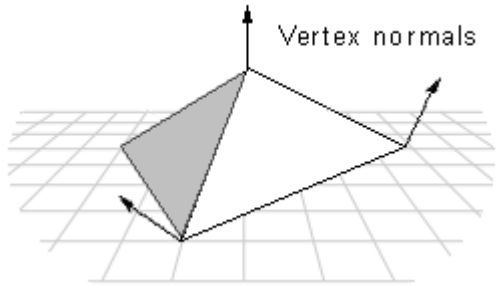


https://en.wikipedia.org/wiki/File:Mesh_overview.svg

Geometric Shapes

- **Vertex** is a point, where two or more curves, lines, or edges meet
- **Triangle** consists of
  - 3 non-collinear vertices interconnected by edges
  - 1 normal vector indicating the front side of the triangle (face) by pointing vertically away from it
  - Additionally normal vectors for the vertices exist





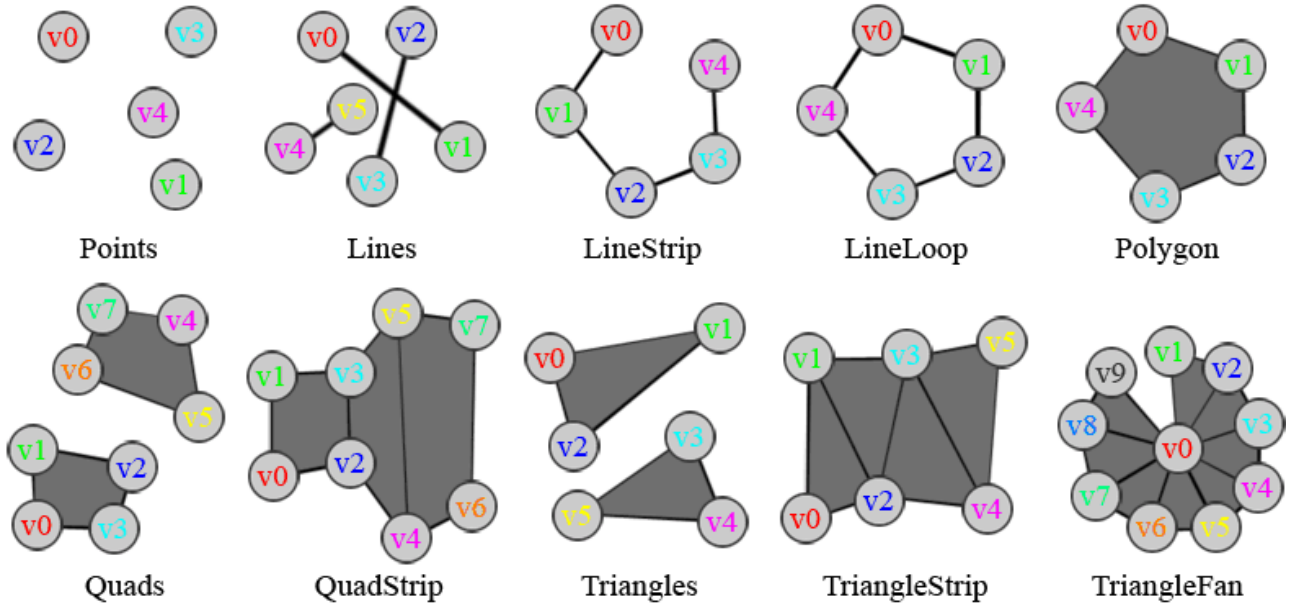
https://docs.microsoft.com/en-us/windows/desktop/direct3d9/face-and-vertex-normal-vectors

- Knowledge of the front side can be important for optimisation (more in culling area of the lecture)

- Face normal vectors and vertex normal vectors can be used for determining the rendered colour of the triangle (more in shading area of the lecture)

Geometric Shapes

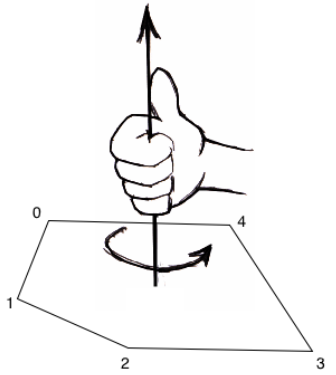
- In graphics APIs typically a type of primitive is set and an ordered list vertices is used to describe the structure of the primitive
- Depending on the order of the list the vertices can be connected by edges and a surface can be generated
- Examples for primitives in modern graphics libraries are
  - Points (not connected)
  - Lines (only two points)
  - Line strips and line loops (no face)
  - Polygons, quads, quad strips, triangles, triangle strips, triangle fans (contain single or multiple faces)
- Can be used for optimised storage e.g. triangles vs. triangle strip



https://de.wikipedia.org/wiki/Grafisches_Primitiv#/media/File:GeometricPrimitiveTypes.png

Geometric Shapes

- Right hand rule is “rule of thumb” to determine the front side (= direction of the normal vector) for a polygon
- Relationship between vertex order and normal vector is a convention not a fixed definition
- Polygons can be stored in different ways in memory or on disc (resulting in a variety of file formats e.g. .obj, .3ds, .dae)
- Examples would be vertex lists and face lists (description where the individual vertices are located in space combined with a description how they are inter-connected)



http://users.monash.edu/~cema/courses/CSE5910/lectureFiles/images/lect8a/frontFacingPolygon.png

Face-Vertex Meshes

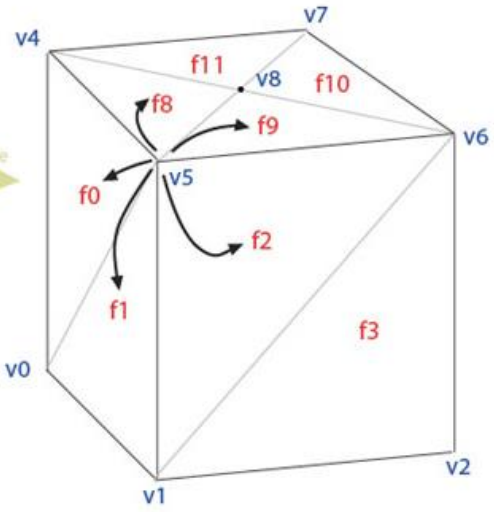
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15

example



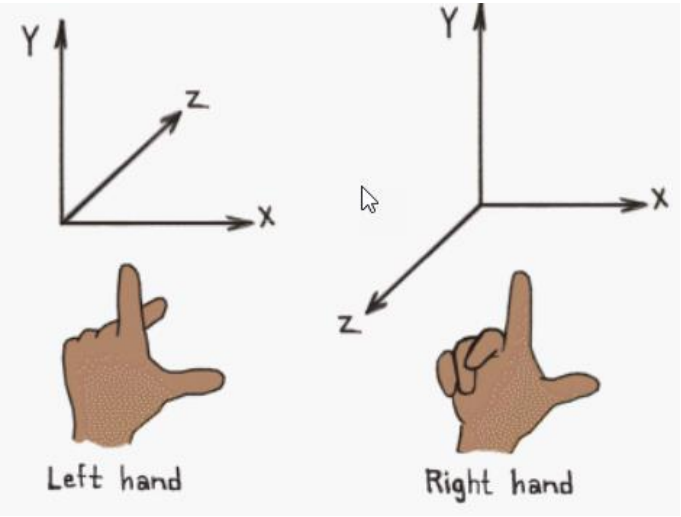
https://upload.wikimedia.org/wikipedia/commons/2/2d/Mesh_fv.jpg

Coordinate Systems

- In computer graphics two-dimensional and three-dimensional coordinate systems are used
- 2D
  - Screen Coordinate System
  - Viewport Coordinate System
- 3D
  - World Coordinate System
    - Objects are located and placed inside this coordinate system
    - Coordinate system of the virtual world
  - Object Coordinate System
    - Local coordinates of each object
    - Each object has own coordinate system
  - Hierarchical Coordinate Systems (used in scene graphs)

Coordinate Systems

- Typically Cartesian (rectilinear) coordinates are used, they have pairwise orthogonal axes with (identical) linear scale
  - Non-cartesian (curvilinear) exist as well e.g. polar/spherical coordinates (angle plus distance), cylindrical
  - If cylindrical or spherical coordinates are used in data representation they are in most cases transformed in Cartesian coordinates for display
- Left-handed vs. right-handed coordinate systems
  - Right-handed
    - X-y plane lies in the screen and Z-axis points to the observer
  - Left-handed
    - X-y plane lies in the screen and Z-axis points away from the observer
  - Unity uses a left-handed coordinate system, OpenGL uses a right-handed coordinate system



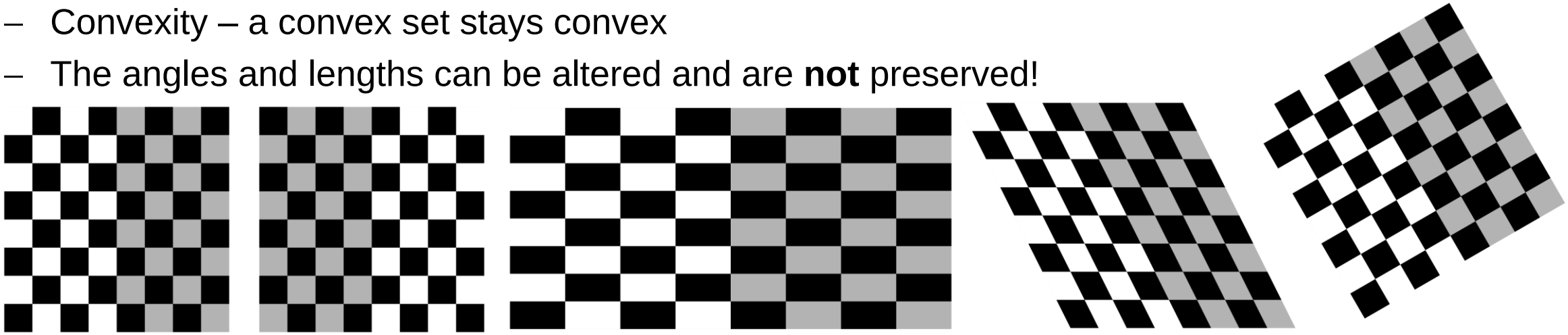
https://www.safaribooksonline.com/library/view/learn-arcore-/9781788830409/03e5338d-02f1-4461-a57a-ef46a976f96b.xhtml

Transformations

- Geometrical transformations are for example translation, scaling, rotation, shearing of objects inside a coordinate system or in-between coordinate systems
- Transformations are needed for arranging objects inside the scene and also for projecting the 3D geometry on a 2D display
- Transformation of the individual vertices is performed to transform the whole polygon, the interconnection between the vertices stays as it is but the object is deformed

Transformations

- For scene layout and manipulation we mainly use affine transformations
- An **affine** transformation preserves the following properties
  - Collinearity – meaning points that are lying on a line before are on a line after transformation
  - The ratio of lengths is kept (e.g. midpoint of a line segment stays the midpoint)
  - Parallelism – parallel lines remain parallel
  - Convexity – a convex set stays convex
  - The angles and lengths can be altered and are **not** preserved!



https://en.wikipedia.org/wiki/Affine_transformation

Identity Reflection Scale Shearing Rotation

Multiplication of Vectors and Matrices

- For transformations we will need to multiply matrices (representing the transformation) and vectors (representing the vertices)
- The number of columns has to be equal to the number of elements in the vector if we want to consider it as a column matrix
- The formula for matrix (M) with vector (v) multiplication is

$$Mv = \begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} m_1v_1 + m_5v_2 + m_9v_3 + m_{13}v_4 \\ m_2v_1 + m_6v_2 + m_{10}v_3 + m_{14}v_4 \\ m_3v_1 + m_7v_2 + m_{11}v_3 + m_{15}v_4 \\ m_4v_1 + m_8v_2 + m_{12}v_3 + m_{16}v_4 \end{pmatrix}$$

- Each element of the row of the matrix is multiplied with the individual vector components and the products are summed up

Matrix-Matrix Product

- For calculating the product of two matrices we consider the second matrix as a set of vectors

$$M = \begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{pmatrix} = \left(\begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{pmatrix} \begin{pmatrix} m_5 \\ m_6 \\ m_7 \\ m_8 \end{pmatrix} \begin{pmatrix} m_9 \\ m_{10} \\ m_{11} \\ m_{12} \end{pmatrix} \begin{pmatrix} m_{13} \\ m_{14} \\ m_{15} \\ m_{16} \end{pmatrix} \right)$$

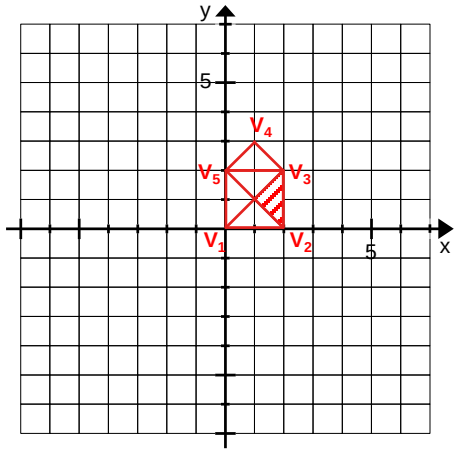
- An example for calculation of the multiplication of the two 3x3 matrices A and B would be

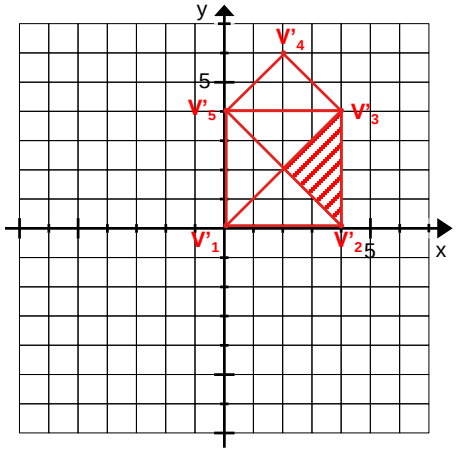
$$AB = \begin{pmatrix} a_1 & a_4 & a_7 \\ a_2 & a_5 & a_8 \\ a_3 & a_6 & a_9 \end{pmatrix} \begin{pmatrix} b_1 & b_4 & b_7 \\ b_2 & b_5 & b_8 \\ b_3 & b_6 & b_9 \end{pmatrix} = \begin{pmatrix} a_1b_1 + a_4b_2 + a_7b_3 & a_1b_4 + a_4b_5 + a_7b_6 & a_1b_7 + a_4b_8 + a_7b_9 \\ a_2b_1 + a_5b_2 + a_8b_3 & a_2b_4 + a_5b_5 + a_8b_6 & a_2b_7 + a_5b_8 + a_8b_9 \\ a_3b_1 + a_6b_2 + a_9b_3 & a_3b_4 + a_6b_5 + a_9b_6 & a_3b_7 + a_6b_8 + a_9b_9 \end{pmatrix}$$

- By using a specific matrix setup we achieve a geometrical transformation of our vertices

Simple 2D Transformations

- Scale an object by 2





- We want to get from

$$v_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} v_2 = \begin{pmatrix} 2 \\ 0 \end{pmatrix} v_3 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} v_4 = \begin{pmatrix} 1 \\ 3 \end{pmatrix} v_5 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

to

$$v'_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} v'_2 = \begin{pmatrix} 4 \\ 0 \end{pmatrix} v'_3 = \begin{pmatrix} 4 \\ 4 \end{pmatrix} v'_4 = \begin{pmatrix} 2 \\ 6 \end{pmatrix} v'_5 = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

with vector matrix multiplication

- We need the following matrix for scaling

$$scale(s_x, s_y) = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2*0+0*0 \\ 0*0+2*0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 2*2+0*0 \\ 0*2+2*0 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 2*2+0*2 \\ 0*2+2*2 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 2*1+0*3 \\ 0*1+2*3 \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 2*0+0*2 \\ 0*0+2*2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

scale V V'

Simple 2D Transformations

- As seen 2D transformation matrices can be used for transformations in 2D

$$Mv = \begin{pmatrix} m_1 & m_3 \\ m_2 & m_4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} m_1 x + m_3 y \\ m_2 x + m_4 y \end{pmatrix}$$

- Scaling and rotation can be applied by multiplying the matrix with a 2D vector

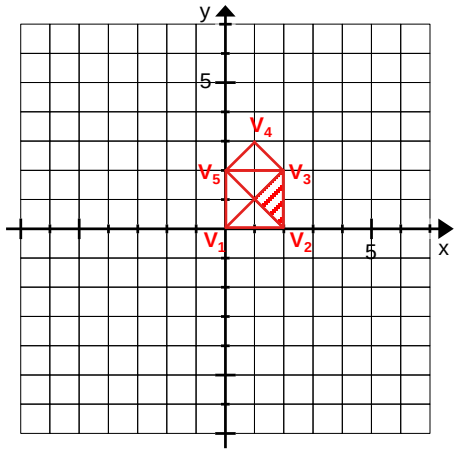
$$scale(s_x, s_y) = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \qquad \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \end{pmatrix}$$

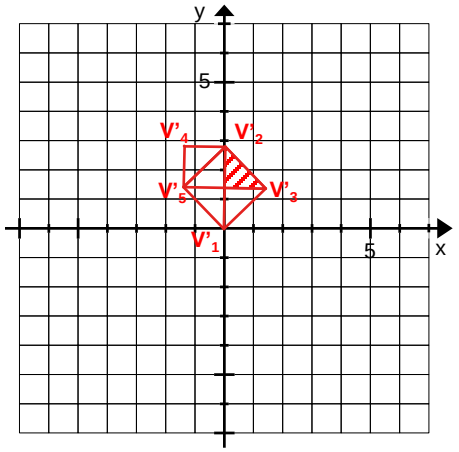
- Rotation around the origin is implemented as well by vector matrix multiplication

$$rotate(\phi) = \begin{pmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{pmatrix} \qquad \begin{pmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} cos\phi x & -sin\phi y \\ sin\phi x & cos\phi y \end{pmatrix}$$

Simple 2D Transformations

- Rotation in 2D space 45° around the origin





- We want to get from

$$v_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} v_2 = \begin{pmatrix} 2 \\ 0 \end{pmatrix} v_3 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} v_4 = \begin{pmatrix} 1 \\ 3 \end{pmatrix} v_5 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

to

$$v'_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} v'_2 = \begin{pmatrix} 1.4 \\ 1.4 \end{pmatrix} v'_3 = \begin{pmatrix} 0 \\ 2.8 \end{pmatrix} v'_4 = \begin{pmatrix} -1.4 \\ 2.8 \end{pmatrix} v'_5 = \begin{pmatrix} -1.4 \\ 1.4 \end{pmatrix}$$

with vector matrix multiplication

- We need the following matrix for rotation

$$rotate(\phi) = \begin{pmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{pmatrix}$$

- Rotating 45° counter clockwise generates the following matrix

$$rotate(45) = \begin{pmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{pmatrix}$$

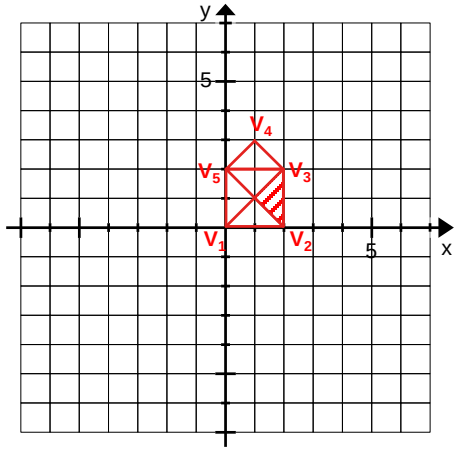
which is then multiplied as we have seen before

Simple 2D Transformations

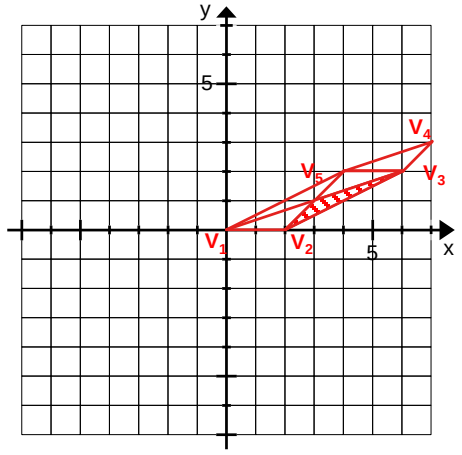
- Shearing and reflection along/around the x and y axis work in a similar way than scaling and rotation

$$identity = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad shear_x(s) = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \qquad shear_y(s) = \begin{pmatrix} 1 & 0 \\ s & 1 \end{pmatrix} \qquad reflect_x = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \qquad reflect_y = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

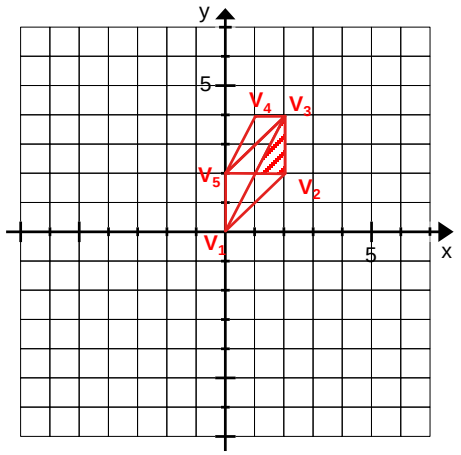
$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \qquad \begin{pmatrix} 1 & s_x \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + s_x y \\ y \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 \\ s_y & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y + s_y x \end{pmatrix} \qquad \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x \\ y \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ -y \end{pmatrix}$$



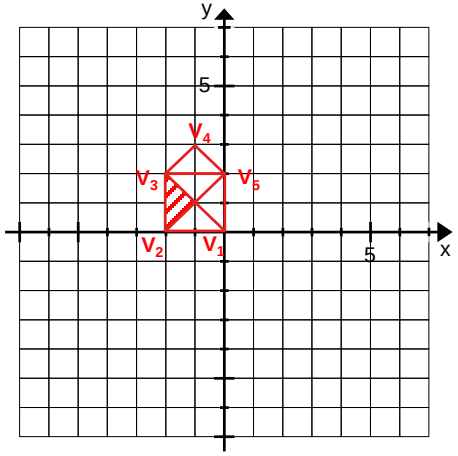
Original



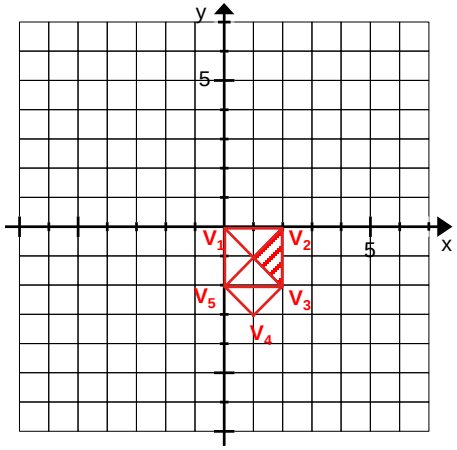
$shear_x(2)$



$shear_y(1)$



$reflect_x$
(reflects x coordinates about y)

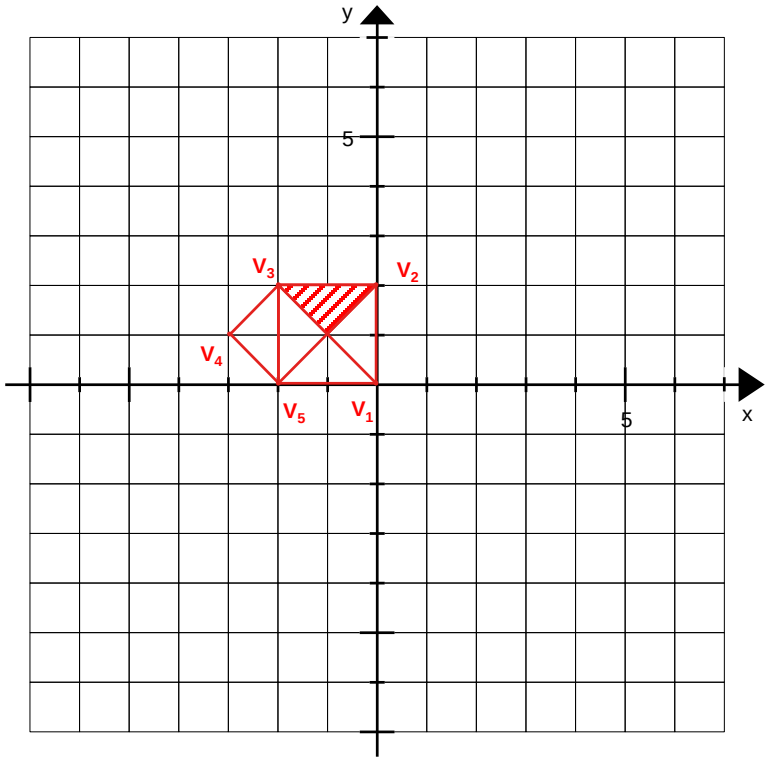


$reflect_y$
(reflects y coordinates about x)

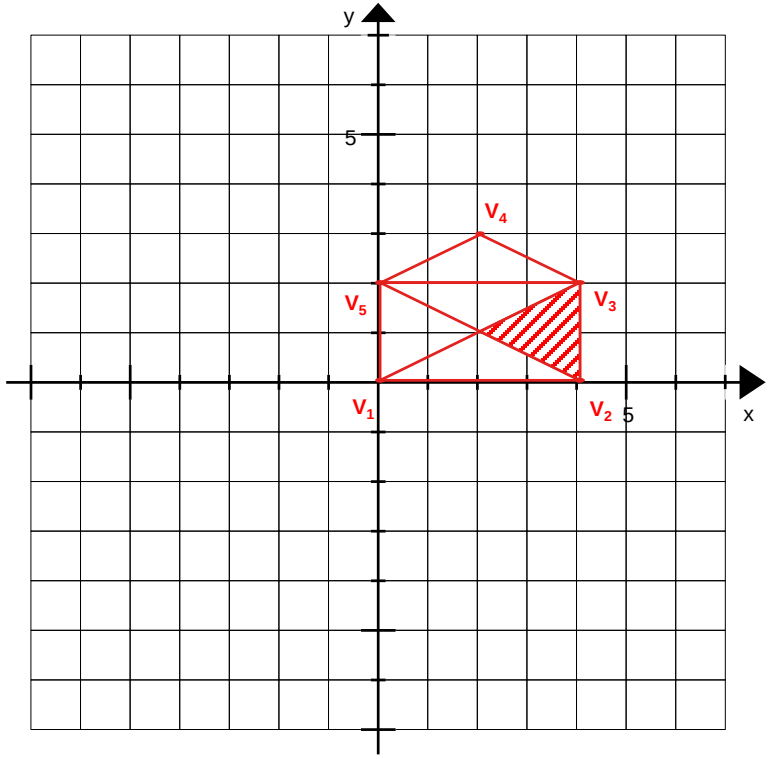
Simple 2D Transformations

- Some example transformations – how do the matrices look like?

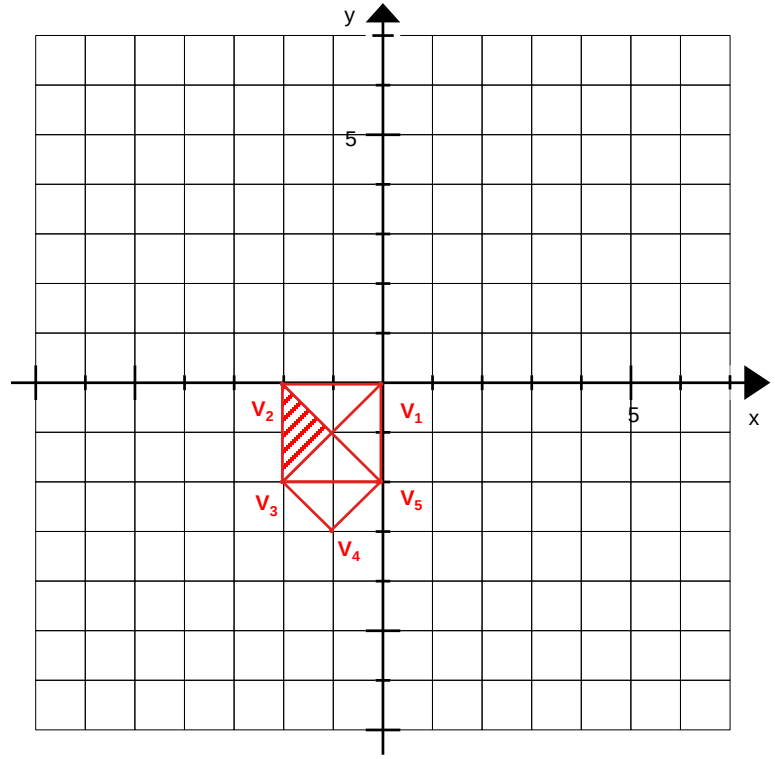
$$M_1 = \begin{pmatrix} cos(90) & -sin(90) \\ sin(90) & cos(90) \end{pmatrix}$$



$$M_2 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$



$$M_3 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$



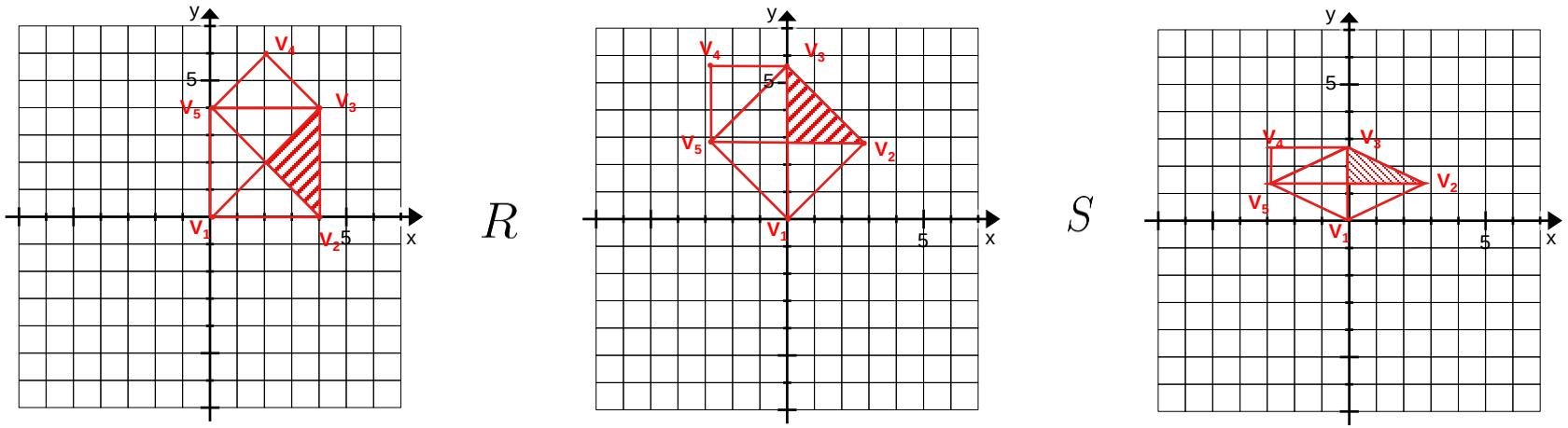
Concatenating Transformations

- To concatenate transformations the matrices are multiplied
- The resulting product applied on a vector provides the same result as applying the individual matrices sequentially
- The order of multiplications plays an **important** role, since matrix multiplication is **not commutative**
- The order of transformations is a very likely source of errors
- Matrix multiplication is **associative**

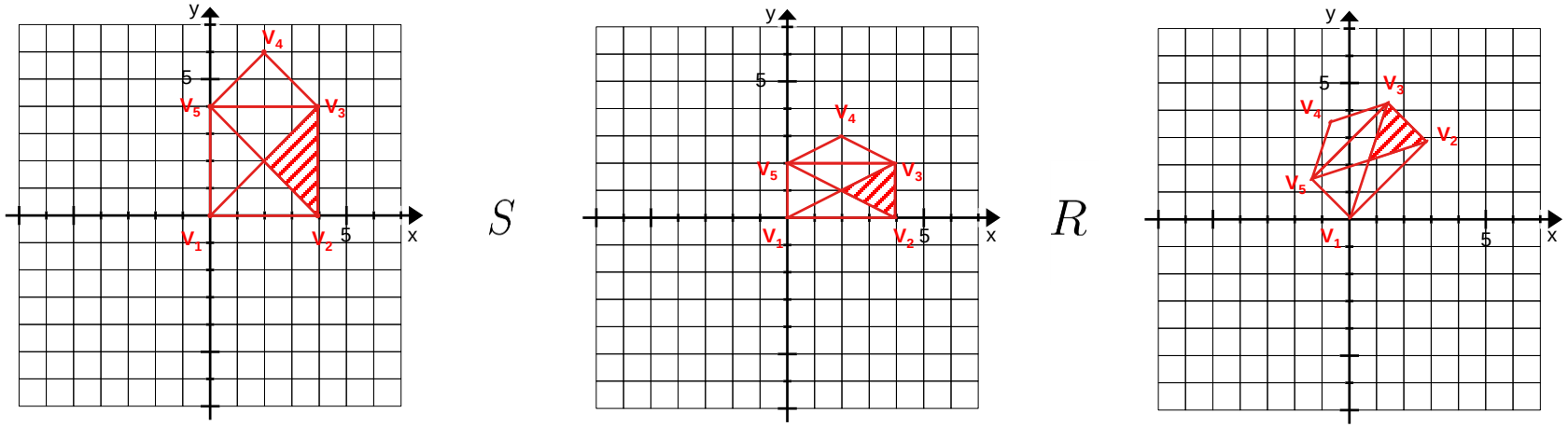
$$v_2 = M_1 v_1$$
$$v_3 = M_2 v_2$$
$$v_3 = M_2(M_1 v_1) = (M_2 M_1) v_1$$
$$v_3 = M_3 v_1$$
$$M_3 = M_2 M_1$$

Concatenating Transformations

- Rotating and scaling an object is different from



scaling and rotating an object



- Our rotation matrix R is defined as

$$R = \begin{pmatrix} cos(45) & -sin(45) \\ sin(45) & cos(45) \end{pmatrix}$$

- Our scaling matrix S is defined as

$$S = \begin{pmatrix} 1 & 0 \\ 0 & 0.5 \end{pmatrix}$$

Homogeneous Coordinates

- To calculate translations we would need to simply add an offset to the given coordinates

$$\begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \end{pmatrix}$$

- Unfortunately this can't be represented with 2D matrices, due to dependencies of the variables, in the way we like to work
- To represent a translation in matrix notation we need an additional coordinate thus we extend the matrix by transforming it into **homogeneous coordinates**

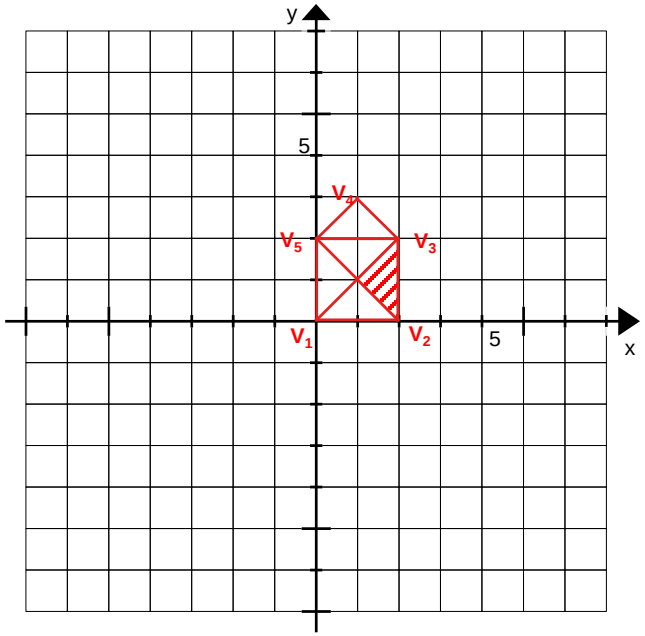
$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \lambda \end{pmatrix} = \begin{pmatrix} v_1/\lambda \\ v_2/\lambda \\ v_3/\lambda \end{pmatrix}$$

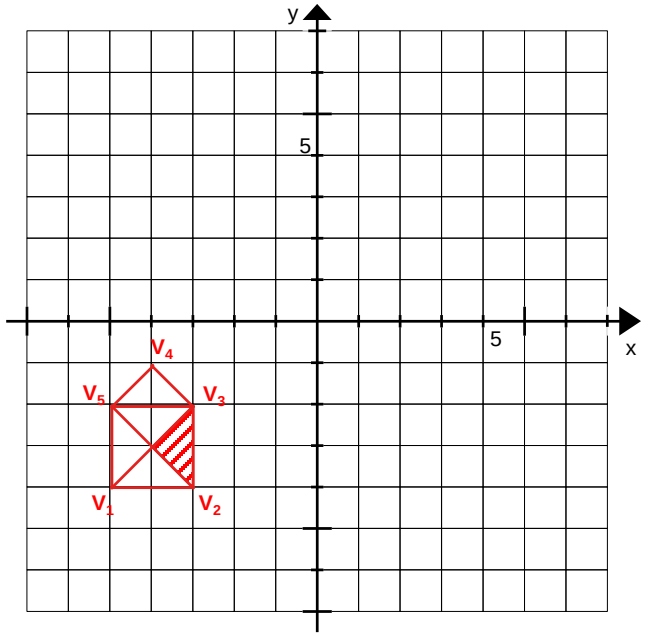
Homogeneous Coordinates

- To perform a translation in 2D space we need to expand our matrices to 3x3

$$translate(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$





$$\begin{pmatrix} 1 & 0 & -5 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x - 5 \\ y - 4 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} (x-5)/1 \\ (y-4)/1 \end{pmatrix}$$

Homogeneous Coordinates

- To multiply our previous matrices with a translation matrix we will also have to extend our already presented matrices with homogeneous coordinates

$$shear_x(s) = \begin{pmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad reflect_x = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad scale(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$shear_y(s) = \begin{pmatrix} 1 & 0 & 0 \\ s & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad reflect_y = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad rotate(\phi) = \begin{pmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Homogeneous Coordinates - Concatenation

- To finally work in 3D space we will need 4x4 matrices to implement translation in 3D space
- Complexity of transformations
  - When broken down a matrix multiplication in homogenous coordinates in our context results in:
    - 9 multiplications
    - 9 additions
- Implemented in hardware which is designed for exactly such tasks (GPU)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & t_x \\ m_{21} & m_{22} & m_{23} & t_y \\ m_{31} & m_{32} & m_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} * x + m_{12} * y + m_{13} * z + t_x \\ m_{21} * x + m_{22} * y + m_{23} * z + t_y \\ m_{31} * x + m_{32} * y + m_{33} * z + t_z \\ 1 \end{pmatrix}$$

Homogeneous Coordinates - Concatenation

- Matrices are associative so they can be combined into one matrix, which is used later on

$$v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A = rotation 90° around x-axis (i.e. Y becomes Z)

B = translation 5 along y-axis

$$(A*B)*v = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 5 \\ 1 \end{pmatrix}$$

- Common approach for transformation
  - Transform the point sets into the origin
  - Perform rotation of the point sets around origin
  - Transform the rotated objects back to the original position with an inversion of the initial transformation matrix

Translation

- Movement in 3D space is one of the most common operations in CG
- It is considered a rigid body transformation since the dimensions of the object stay identical
- Translation of a point p

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$p(x, y, z)$$

with the vector v

$$v(t_x, t_y, t_z)$$

will lead to the resulting point

$p(x', y', z')$ with $x' = x + t_x, y' = y + t_y, z' = z + t_z$

Scaling

- Increasing or decreasing the size of polygon
- Scaling from the origin

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- With uniform or isotropic scaling $s_x = s_y = s_z$
- Otherwise scaling is called anamorphic, unisotropic or non-uniform scaling
- When we have a scale factor larger than one we speak of dilation or enlargement if it is smaller we speak of contraction
- Could be problematic if the object is not at the origin

- Scaling is performed from a fixed point

Rotation

- Rotation around the x-axis

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\phi & -sin\phi & 0 \\ 0 & sin\phi & cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

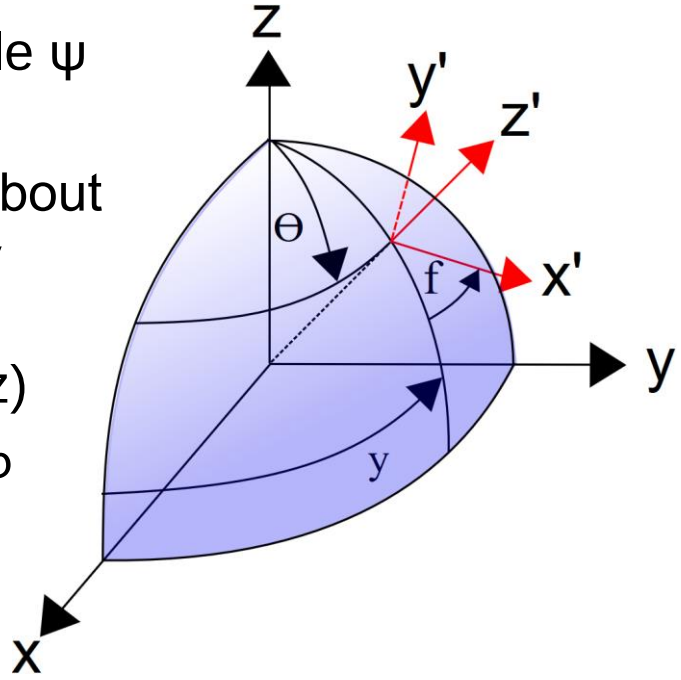
- Rotation around the y-axis

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} cos\phi & 0 & sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -sin\phi & 0 & cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- Rotation around the z-axis

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} cos\phi & -sin\phi & 0 & 0 \\ sin\phi & cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

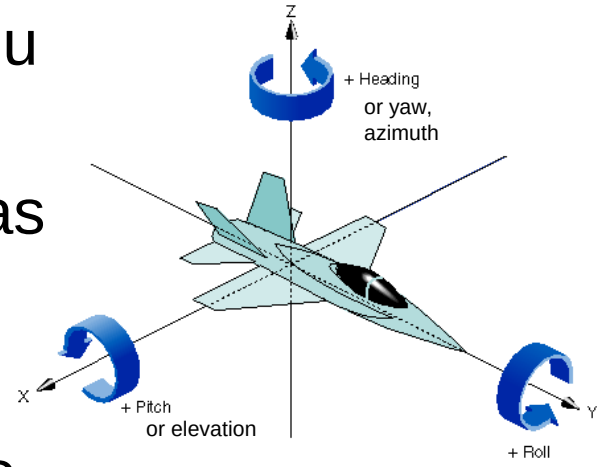
- Basic rotations can be performed by sequentially multiplying rotation matrices
- One common order is the Roe convention
  - (i) rotation through angle ψ about z-axis
  - (ii) rotation of angle θ about the new y-axis (already rotated itself due to the first rotation about z)
  - (iii) second rotation of φ about the now-tilted z-axis



http://www.continuummechanics.org/images/rotationmatrix/roe_coord_xform.svg

Quaternions

- The problem with a given set of rotation operations is that you can end up in a so called Gimbal lock
- If pitch and yaw are aligned, roll operates on the same axis as yaw

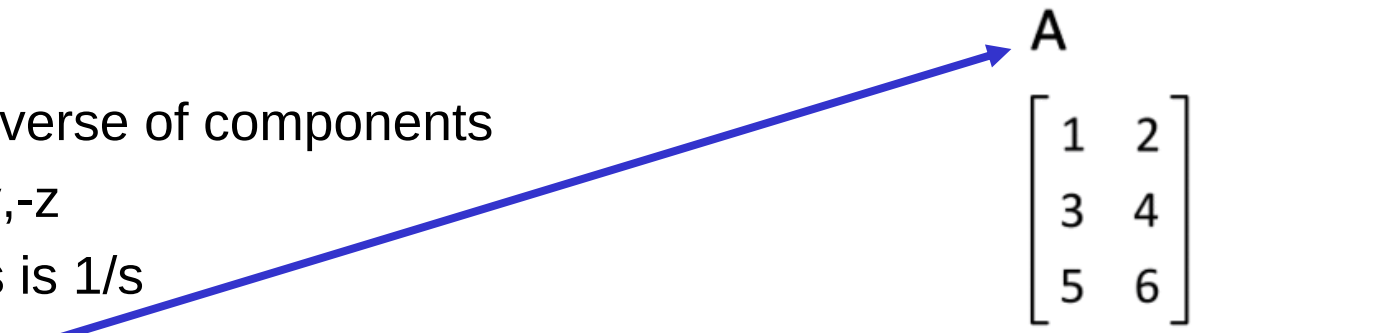


- To solve this issue often quaternions are used, an alternative way to describe a rotation in space
- A quaternion consists of a rotation angle w and a normalised vector describing the axis of rotation

https://upload.wikimedia.org/wikipedia/commons/5/5c/Gimbal_lock_airplane.gif

Normal Vectors and Inverse Transformations

- Normal Vectors
  - When transformed of a shape takes place Normal vectors might not be perpendicular to our shape
  - We would have to transform the Normal vectors as well to have them perpendicular again
- Inverse Matrices
  - Can be easily computed by the inverse of components
  - Inverse translation of x,y,z is -x,-y,-z
  - Inverse scale with scaling factor s is 1/s
  - Inverse rotation is the transpose

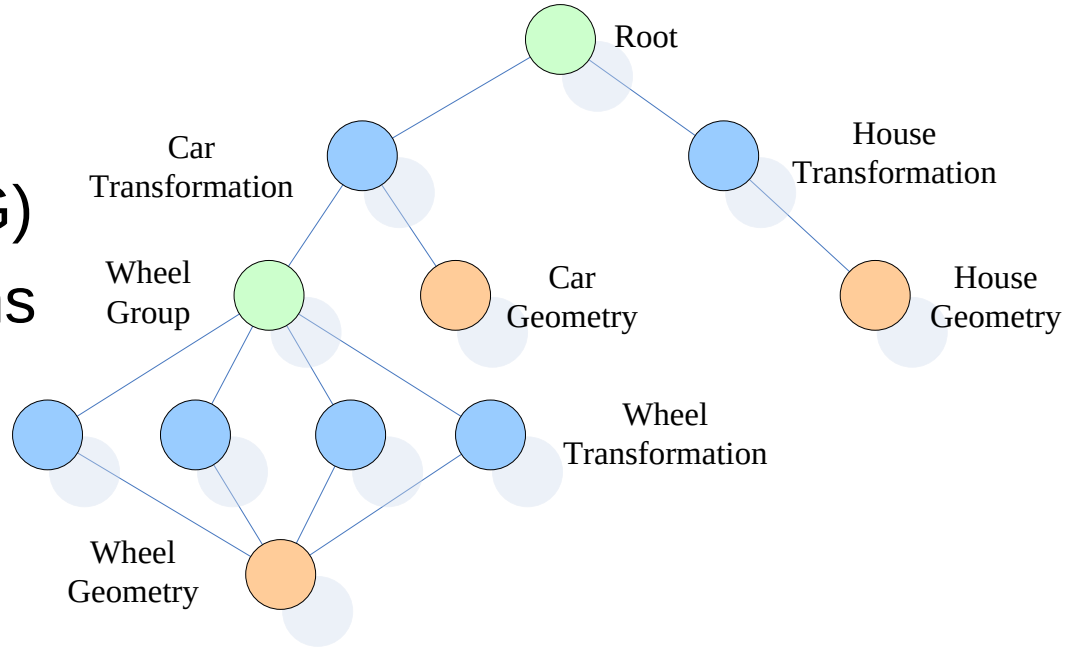


https://de.wikipedia.org/wiki/Transponierte_Matrix#/media/File:Matrix_transpose.gif

- Composed matrices are inverted by using the reverse order of multiplication (inversion of their components)
  - $M=M_1M_2\ldots M_n \rightarrow M^{-1}=M_n^{-1}\ldots M_2^{-1}M_1^{-1}$

Scene Graphs

- Used to provide a higher abstraction layer for graphics programming compared to low-level APIs (used as well in file formats and game engines)
- Scene graphs are used in 3 key areas to support graphics programming
  - Object representation
  - Interactivity
  - Architecture
- Organised as a Directed Acyclic Graph (DAG)
- Follow the concepts of nested transformations
- Structure
  - Internal nodes are used for hierarchy building
  - Leaf nodes contain geometry
  - Attributes are typically inherited top – down
  - Parent nodes can have more than a single child node



Scene Graphs

- Hierarchy
  - Structure lends itself naturally to intuitive spatial organisation
- Culling
  - Removing everything from a scene that does not contribute to the final image
  - Frustum culling compares object boundaries with viewing frustum
  - Occlusion culling checks whether objects are hidden behind others (not often implemented in scene graphs)
- Bounding volume hierarchy
  - Parent nodes bounding volume contains all volumes of child nodes
  - If parent is culled away, all of its children are culled away
- File I/O
  - Reading and writing of 3D models from disk
  - Internal data structure allows the application to easily manipulate dynamic 3D data

Scene Graphs

- Traversal of the scene graph
  - Visiting of each node of the graph consists normally of 3 phases
  - Update or application
    - Update traversal allows modifications of the scene graph
    - Updates are applied either directly by the application or with callback functions assigned to nodes within the scene graph
    - Used for animation or input processing
  - Cull
    - The scene graph tests the bounding volumes of all nodes
    - If leaf node is within the view, a reference to the leaf node geometry is added to a final rendering list
    - List is sorted by opaque versus translucent, and translucent geometry is further sorted by depth
  - Draw or render
    - List of geometry created during the cull traversal is traversed and low-level graphics API is called to render that geometry

Bibliography

- James F. Blinn, “Jim Blinn’s Corner: Nested transformations and blobby man“, *IEEE Computer Graphics and Applications,* 1987*, 7*, pages 65-69
- Avi Bar-Zeev, “Scenegraphs: Past, Present and Future“, last visited February ‘19 http://www.realityprime.com/articles/scenegraphs-past-present-and-future

- Scene Graphs - Example applications
  - Open Inventor (https://www.openinventor.com/)
  - OpenSceneGraph (http://www.openscenegraph.org/)
  - OpenSG (https://sourceforge.net/projects/opensg/)
  - Unity (https://unity3d.com)
  - Unreal (https://www.unrealengine.com/)

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Computer Graphics

Projections, Clipping, and Culling

FH-Prof. Dr. techn. Dipl.-Inf. (FH) Christoph Anthes, MSc

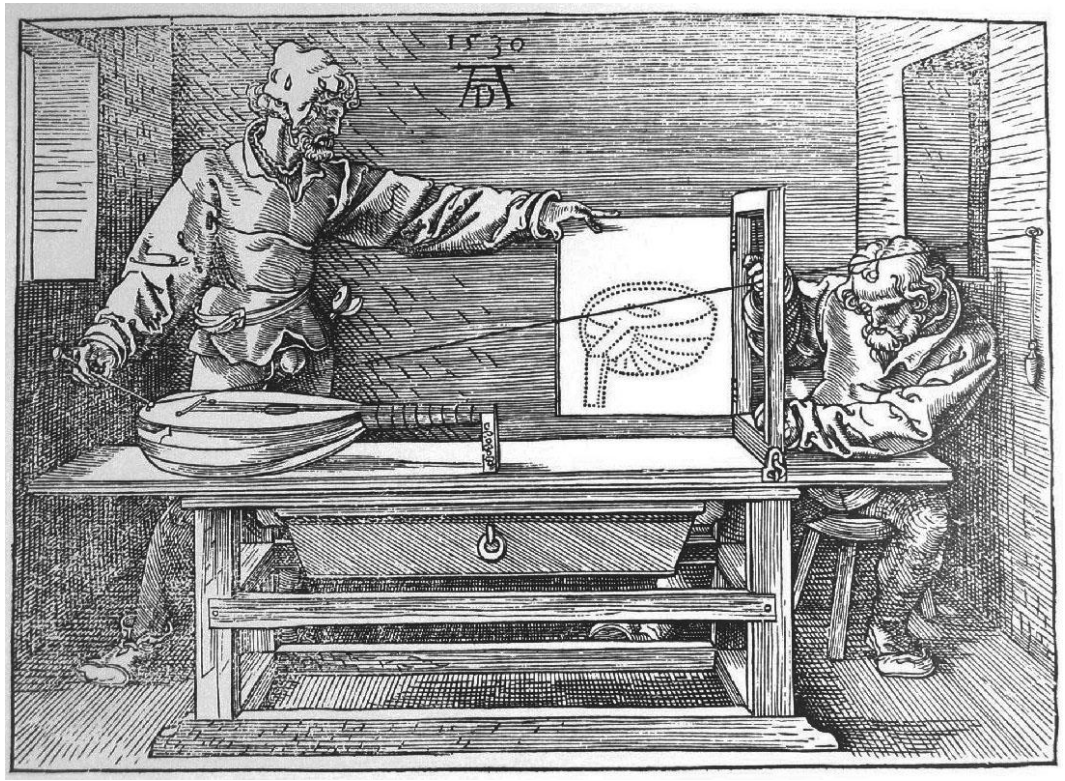
Professor of Augmented and Virtual Reality

Overview

- Projections
  - Frustum
  - Perspective Projection
  - Parallel Projection
- Clipping
  - Cohen-Sutherland Algorithm
  - Sutherland-Hodgman Algorithm
- Culling Techniques
  - Frustum Culling
  - Backface Culling
  - Occlusion Culling

Projections

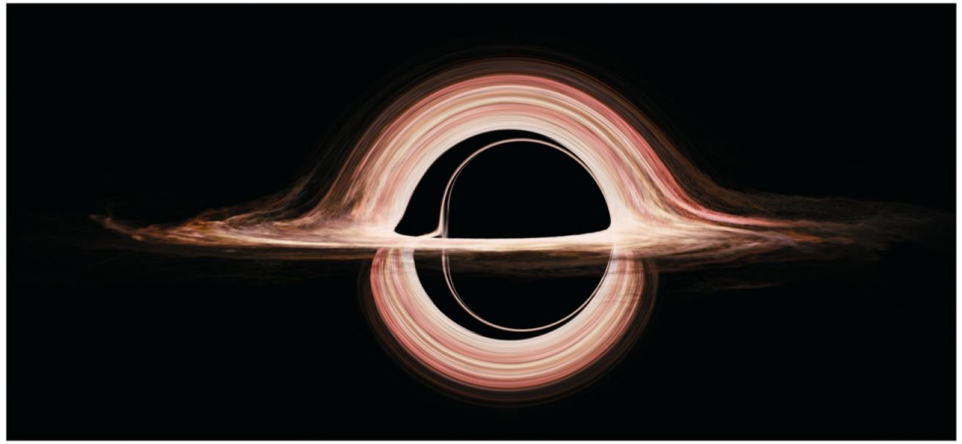
- In order to display the modelled and transformed primitives we need to project the three-dimensional data on a two-dimensional plane
- A common problem from the domain of art
- Projections take place from **world space** (where the scene is modelled) into **screen space** (the actual display on a screen)
- Elements required are the **projection plane** (e.g. later on shown on a display) the **centre of projection** (e.g. the eye of the user/observer) and the **objects** (e.g. consisting of primitives) arranged in the environment



Carlbom, I. & Paciorek, J. Planar Geometric Projections and Viewing Transformations ACM Computing Surveys, Association for Computing Machinery (ACM), 1978, 10, 465-50

Projections

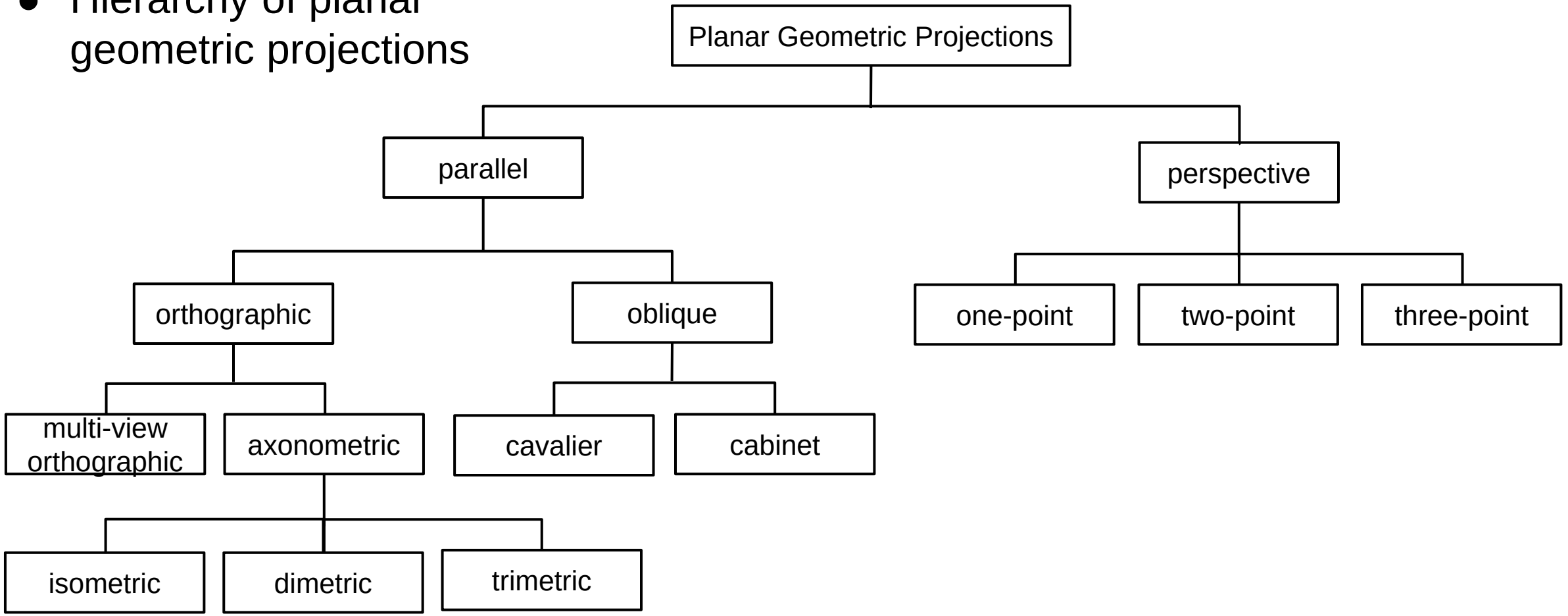
- Planar geometric projections
  - Obtained by passing lines called projectors
  - One through each point of the object
  - Finding the image formed by the intersections of these projectors with a plane of projection
- In CG most common type of projections are planar geometric projections
- Alternatives could
  - Use different projection area than plane (e.g. sphere)
  - Use different mapping than linear projectors (e.g. backwards ray-tracing to rays that propagate on a curved space time)
- Non-planar and non-geometric projections are used for example excessively in maps



James, O.; von Tunzelmann, E.; Franklin, P. & Thorne, K. S. Gravitational lensing by spinning black holes in astrophysics, and in the movie Interstellar *Classical and Quantum Gravity, IOP Publishing,* 2015, *32*, 065001

Projections

- Hierarchy of planar geometric projections

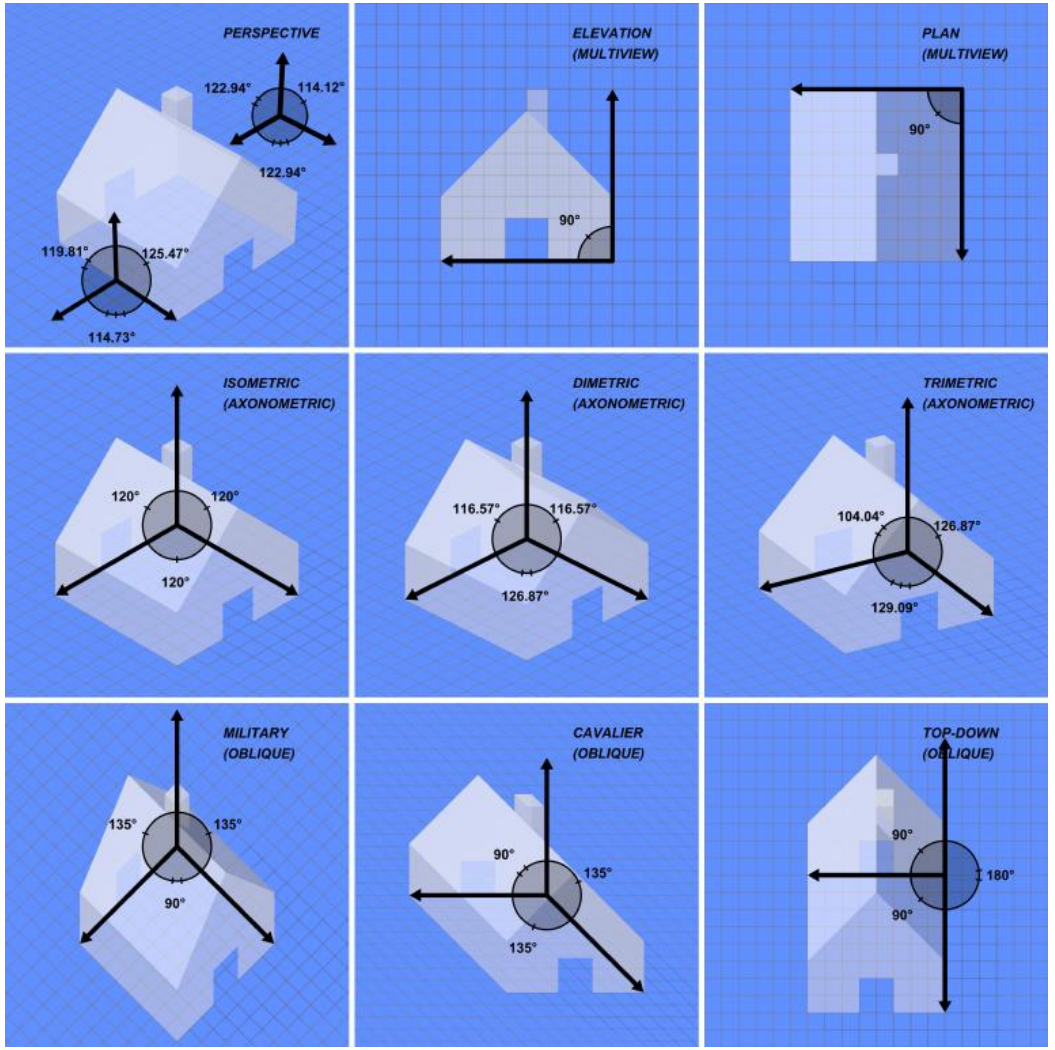


Carlbom, I. & Paciorek, J. Planar Geometric Projections and Viewing Transformations ACM Computing Surveys, Association for Computing Machinery (ACM), 1978, 10, 465-50

Projections

- Depending on the application area different possibilities of projection exist
- The most common option in computer graphics is the perspective projection

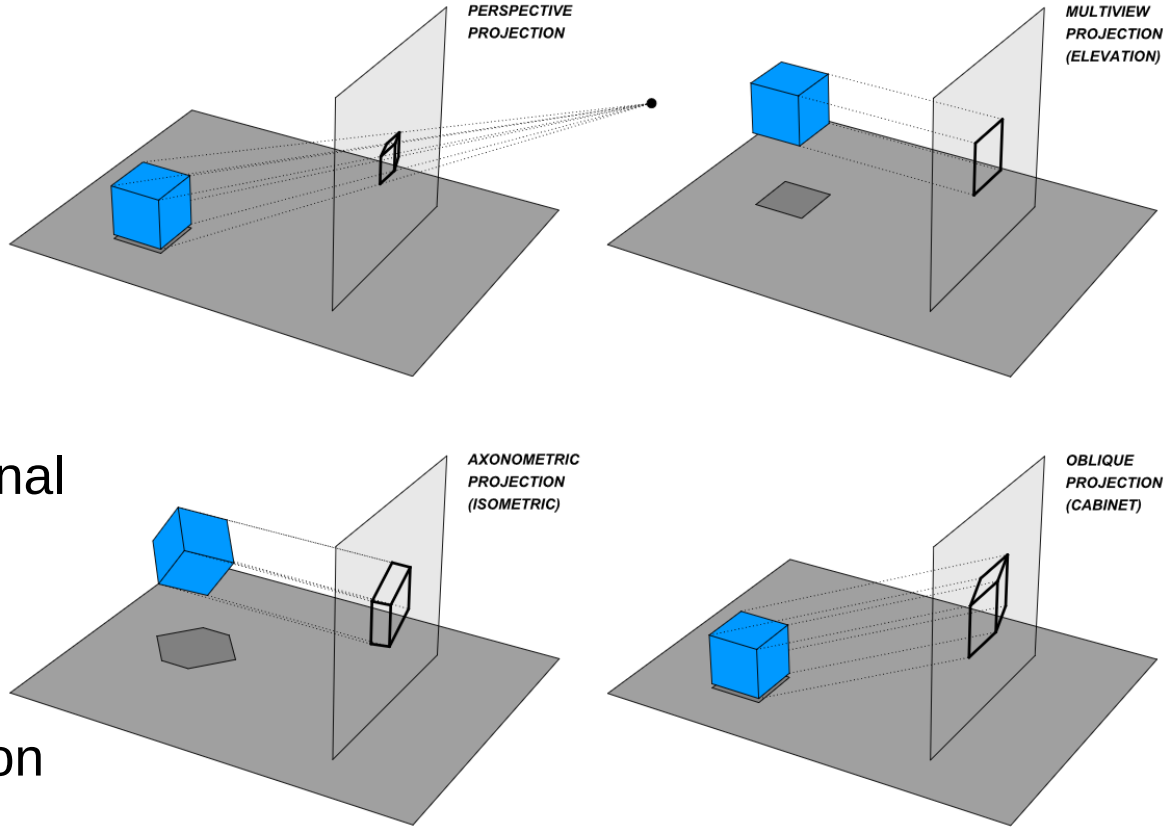
- Variety of application areas for different projections
  - Architecture
  - Games
  - 3D Design
  - Maps
  - Information Visualisation



https://en.wikipedia.org/wiki/Graphical_projection#/media/File:Graphical_projection_comparison.png

Projections

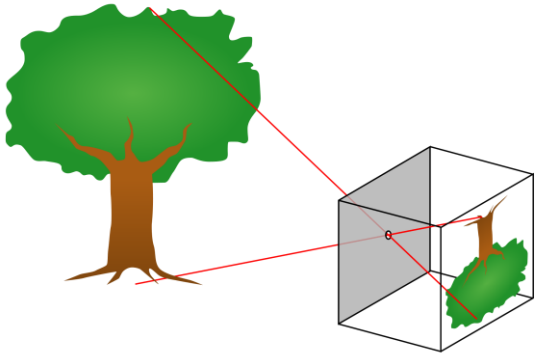
- These different projection types break down into two big categories and multiple subcategories
  - Perspective projection
    - Fixed viewpoint with given coordinates
    - Centre of projection is finite
    - Perspective distortion is result of perspective projection
    - Distant objects are downscaled proportional to the distance on the projection plane
  - Orthographic or parallel projection
    - In parallel projection the parallel lines of the object are also parallel in the projection
    - If has an infinite focal length and the centre of projection is in infinity



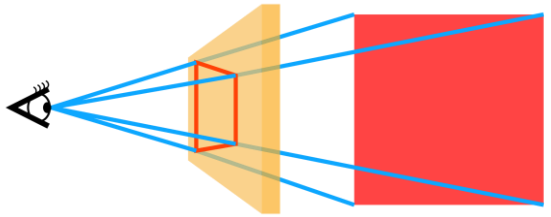
https://upload.wikimedia.org/wikipedia/commons/4/4b/Various_projections_of_cube_above_plane.svg

Projections

- In photography, we usually have the centre of projection in-between the object and the image plane
- Image on film or sensor is recorded upside down
- In CG perspective projection, the image plane is in front of the camera
- Image in CG is not flipped
- In general the same mechanisms and rules apply for a perspective projection in graphics as in film



https://en.wikipedia.org/wiki/Pinhole_camera_model#/media/File:Pinhole-camera.svg

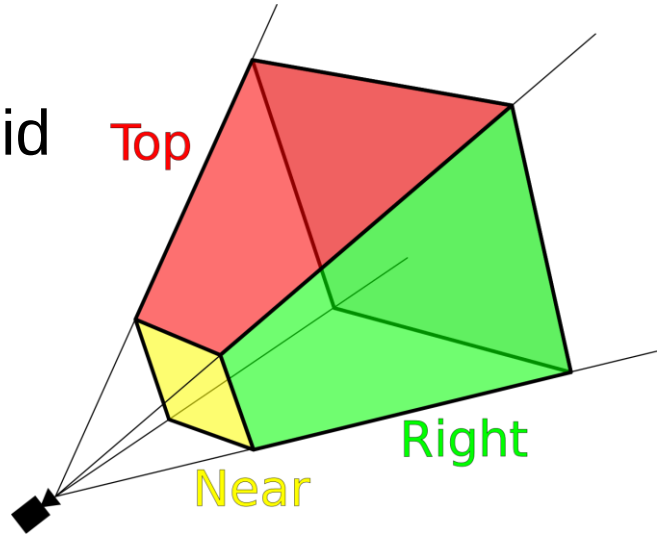


https://en.wikipedia.org/wiki/Perspective_(graphical)#/media/File:Drawing_Square_in_Perspective_2.svg

- Virtual camera (centre of projection) has to be placed in world space as well as the objects in the scene
- We will first look at the projection and then alter the camera position and orientation

Frustum

- The Frustum describes a volume, which content is projected for rendering areas outside the frustum are not to be rendered
- With perspective projection the frustum is a truncated pyramid
- Top, bottom, right and left planes restrict the area as well as the near and the far clipping plane
- Alternative naming convention for clipping planes
  - *Near Clipping Plane (hither)*
  - *Far Clipping Plane (yon)*
- With parallel projection we use a cuboid instead of a frustum with a surface matching the shape of the display plane



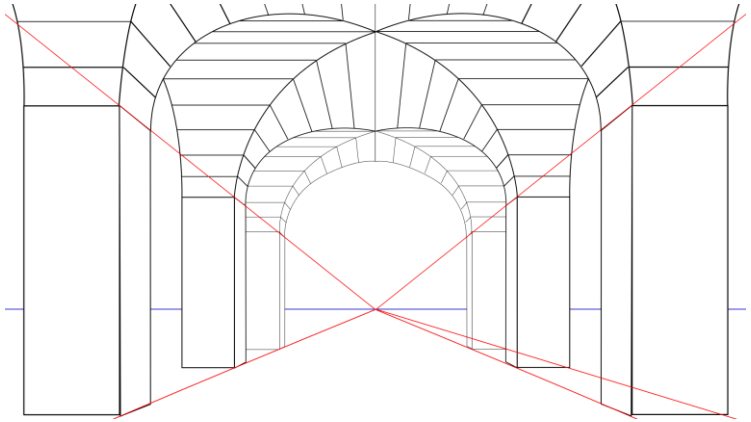
Perspective Projection

- Perspective projection illustrates general appearance of an object as it would be seen by the eye
- Objects further away from the centre of projection appear smaller, objects closer appear larger – the image size is inversely proportional to the distance
- The projection of a point is not unique. Points lying on the same line of projection are projected at the same location in the plane
- When a line or surface is parallel to the projection plane the effect is scaling
- Perspective projection is thus not an affine transformation anymore

Perspective Projection

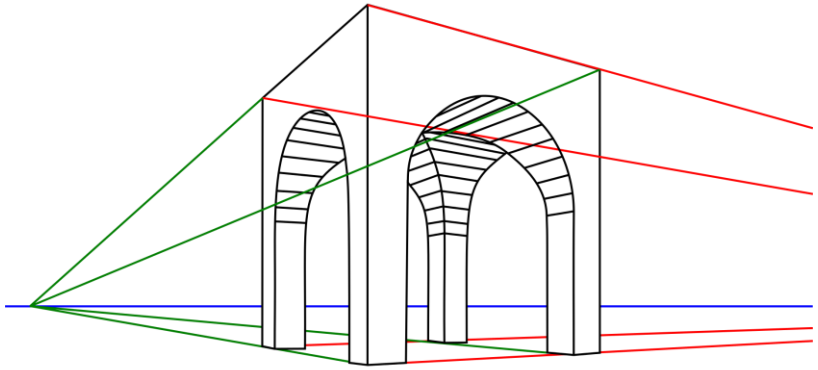
- Typically a single vanishing point is used which lies along the viewing axis of the camera
- Up to 3 vanishing points can be used for multi-point perspective, projection depending on the used axes

One-point perspective



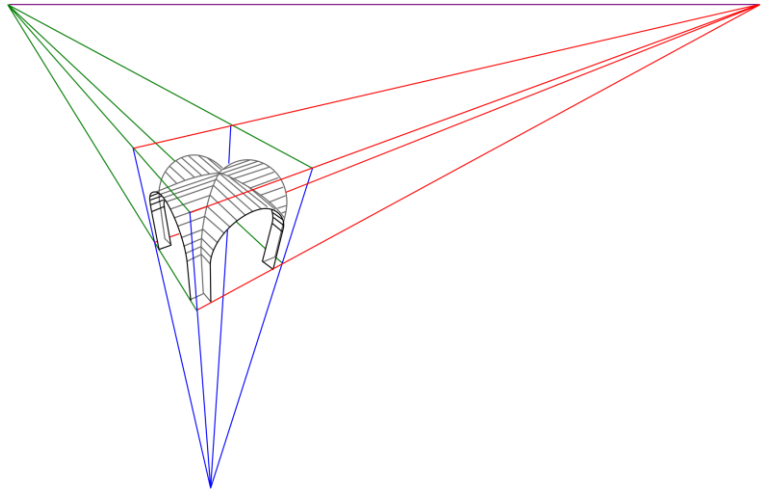
https://de.wikipedia.org/wiki/Fluchtpunkt#/media/File:1ptPerspective.svg

Two-point perspective



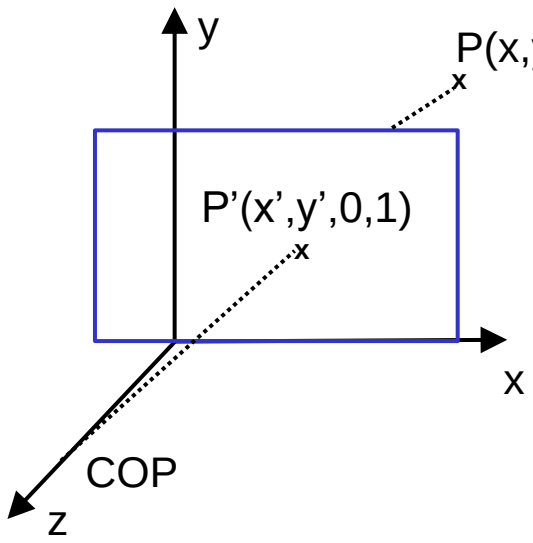
https://de.wikipedia.org/wiki/Fluchtpunkt#/media/File:2-punktperspektive.svg

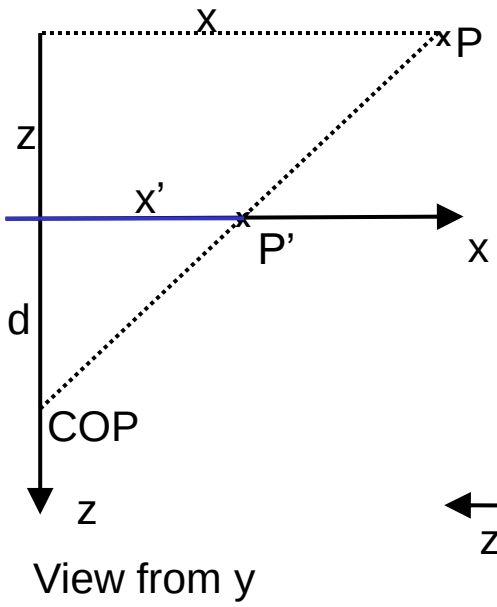
Three-point perspective



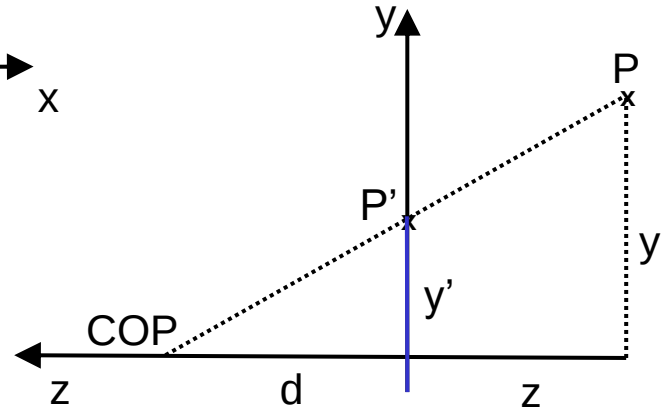
https://de.wikipedia.org/wiki/Fluchtpunkt#/media/File:3-point_perspective_1-px-line.svg

Perspective Projection - Simplified Model





View from y



View from x

$$x' = \frac{d * x}{z + d}$$

$$y' = \frac{d * y}{z + d}$$

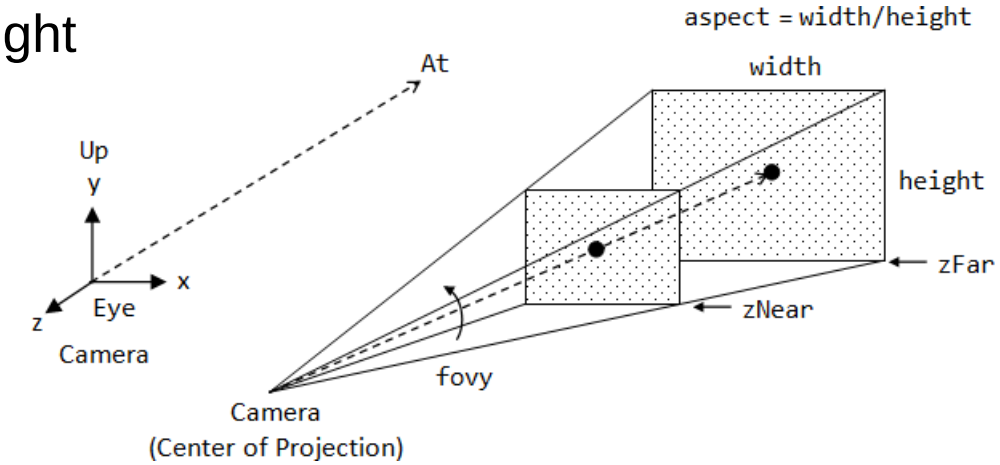
$$z' = 0$$

- Practical application of the theorem on intersecting lines
- We represent P'(x',y',z') in homogeneous coordinates

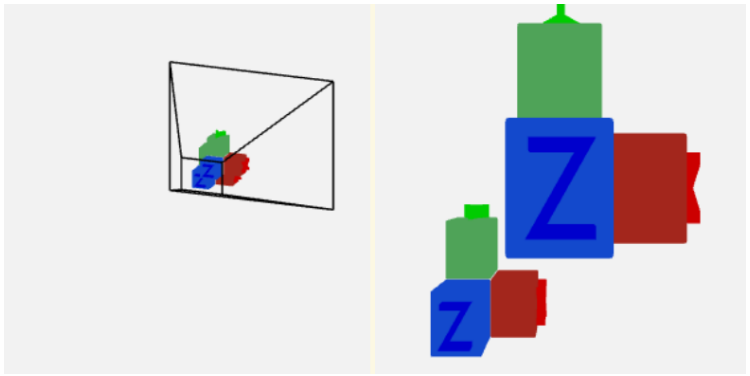
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} d * x \\ d * y \\ 0 \\ z + d \end{pmatrix}$$

Perspective Projection

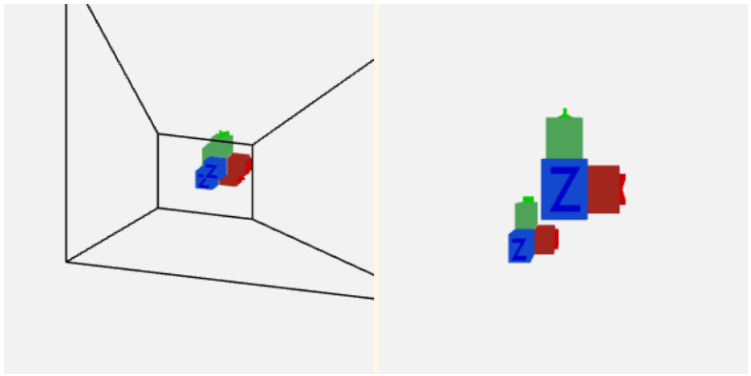
- Other relevant parameters for setting up a frustum for perspective projection
  - Aspect ratio is the width of the area divided by height of area (e.g. 4:3, 16:9)
  - Field of View (FOV) is an angle describing the opening angle of the camera (can be defined as diagonal FOV or horizontal and/or vertical FOV)
- How do changes of FOV and aspect ratio affect our image with constant resolution?



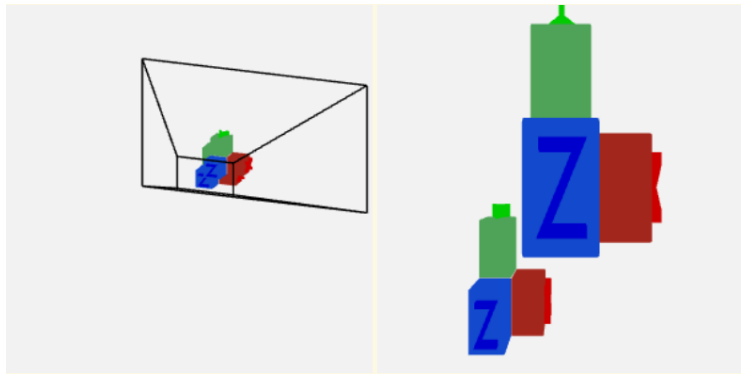
http://www.c-jump.com/bcc/common/Talk3/Math/Matrices/W01_0210_perspective_projectio.htm



FOVY = 40; Aspect 4:3



FOVY = 80; Aspect 4:3

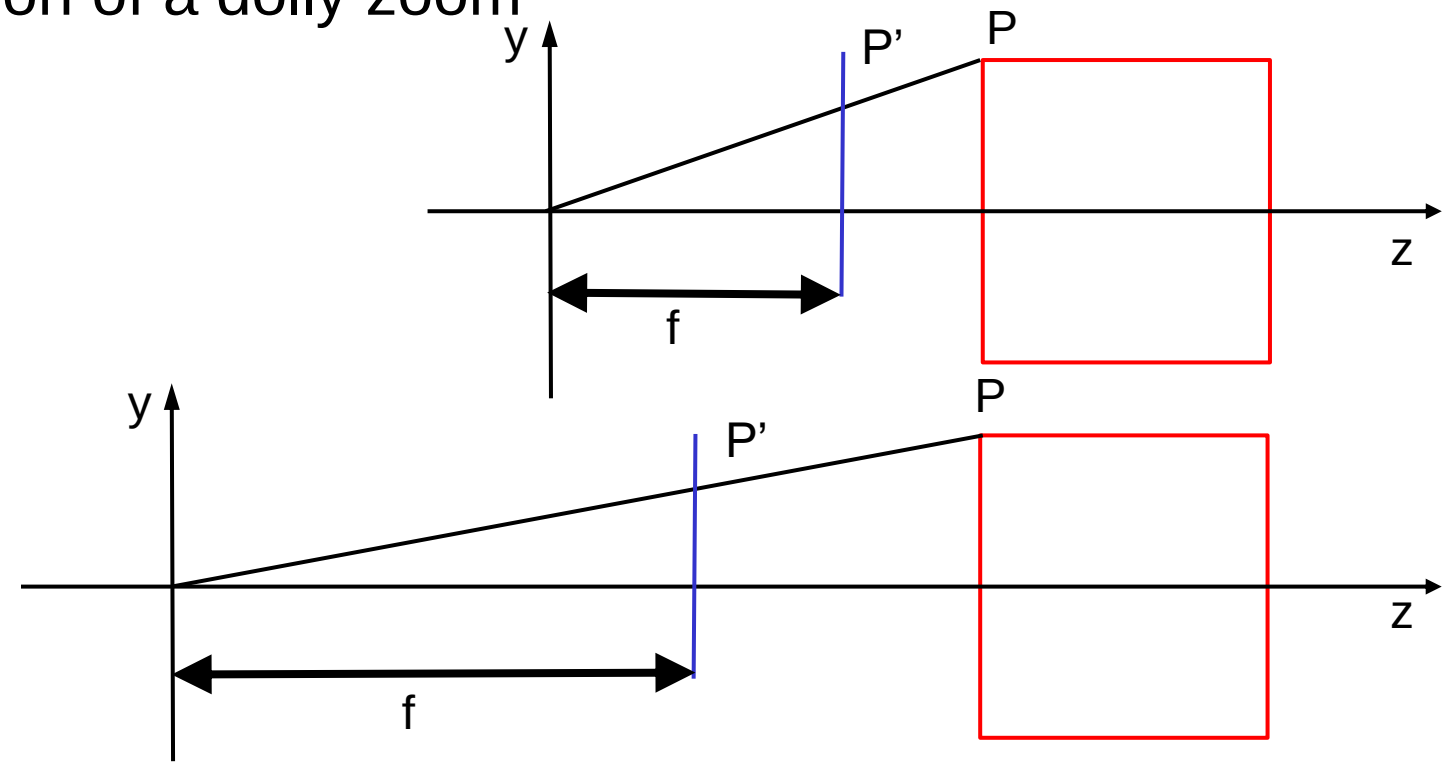


FOVY = 40; Aspect 16:9

Dolly Zoom (Clip from Jaws 1975)

Dolly Zoom

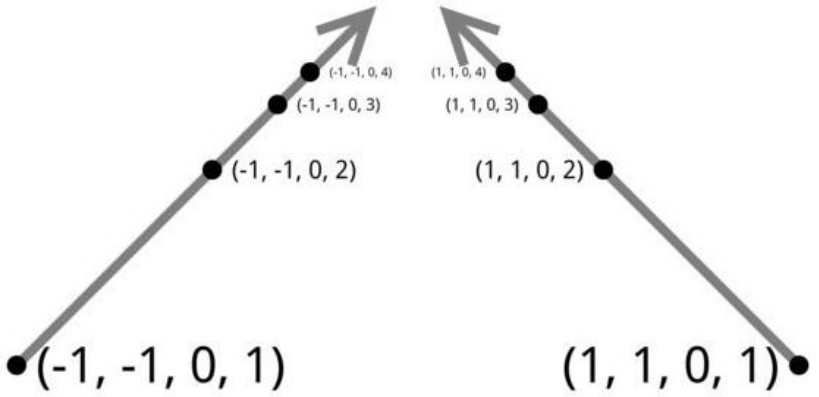
- Implementation of a dolly zoom



- Change of focal length f (denoted as d in our previous model) to keep one object in focus and simultaneously moving the camera away from this object

Perspective Division

- Additionally to the projection we need the perspective division realised by the 4th component the w
- The perspective division is used to transform homogeneous coordinates back into Cartesian coordinates



https://www.learnopengles.com/wordpress/wp-content/uploads/2012/10/divideByW.jpg

- The last coordinate of the vector in homogeneous coordinates is used to divide the individual components of the vector

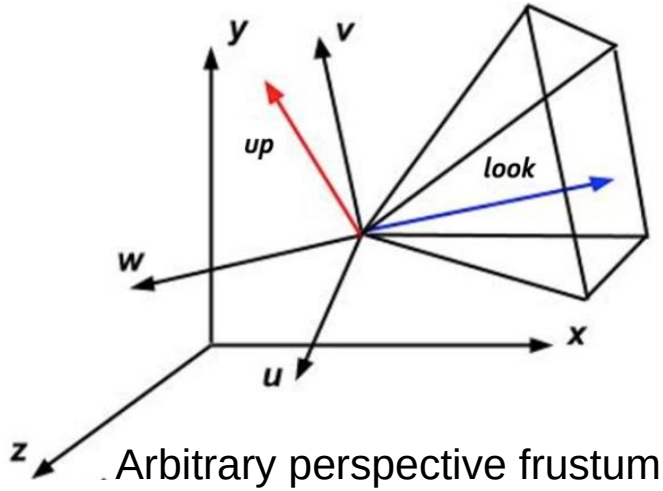
Orthographic Projection - Simplified Model

- Projection is parallel to a coordinate plane - parallel lines project to parallel lines so the size does not change with distance from the camera
- Often used in a multi-view combination sometimes together with an overview (perspective again)
- Advantages are no distortions and use for measurements
- Attempt to represent its metric properties
- Most simple representation would be omitting the z coordinate

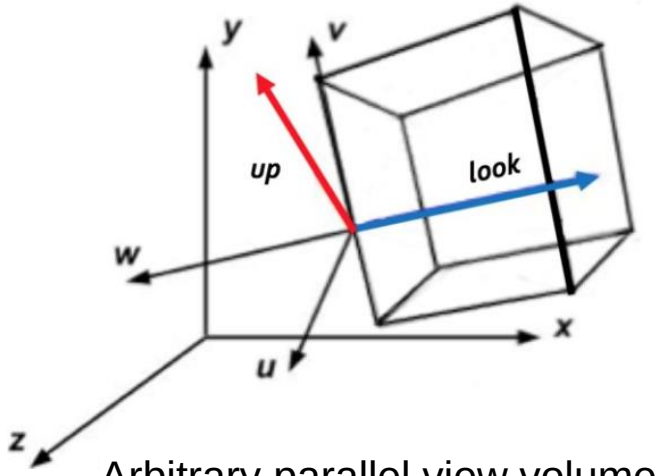
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Changing the View

- 2 alternatives exist
  - Keeping the centre of projection and the projection plane (or viewing plane (VP)) constant but transforming the object(s)
  - Keeping the object transformation constant and changing the centre of projection and projection plane
- For changing the view we need the model of a camera (synthetic camera model) consisting of
  - Position (or view reference point (VRP) the position in world space, where the camera is located)
  - Up vector (or view up vector (VUP) originating in the camera position)
  - Look vector (or view plane normal vector) is perpendicular to the up vector of the camera



Arbitrary perspective frustum



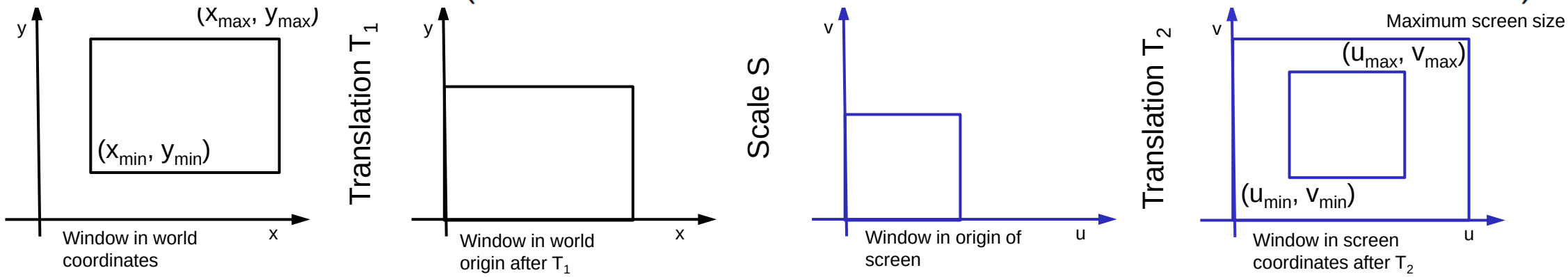
Arbitrary parallel view volume

Brown University, cs123 INTRODUCTION TO COMPUTER GRAPHICS, 2015 course notes

From Image Coordinates to Display Coordinates

- Transformation in 2D space consisting of three simple steps (Window-Viewport-Transformation)
  - 1.) Translation of rendered image to the origin of the screen
  - 2.) Scaling of the image to the desired size in display coordinates
  - 3.) Translation in the final display coordinates

$$M_{WS} = T_2 S T_1 = \begin{pmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & -x_{min}\frac{u_{max}-u_{min}}{x_{max}-x_{min}} + u_{min} \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & -y_{min}\frac{v_{max}-v_{min}}{y_{max}-y_{min}} + v_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

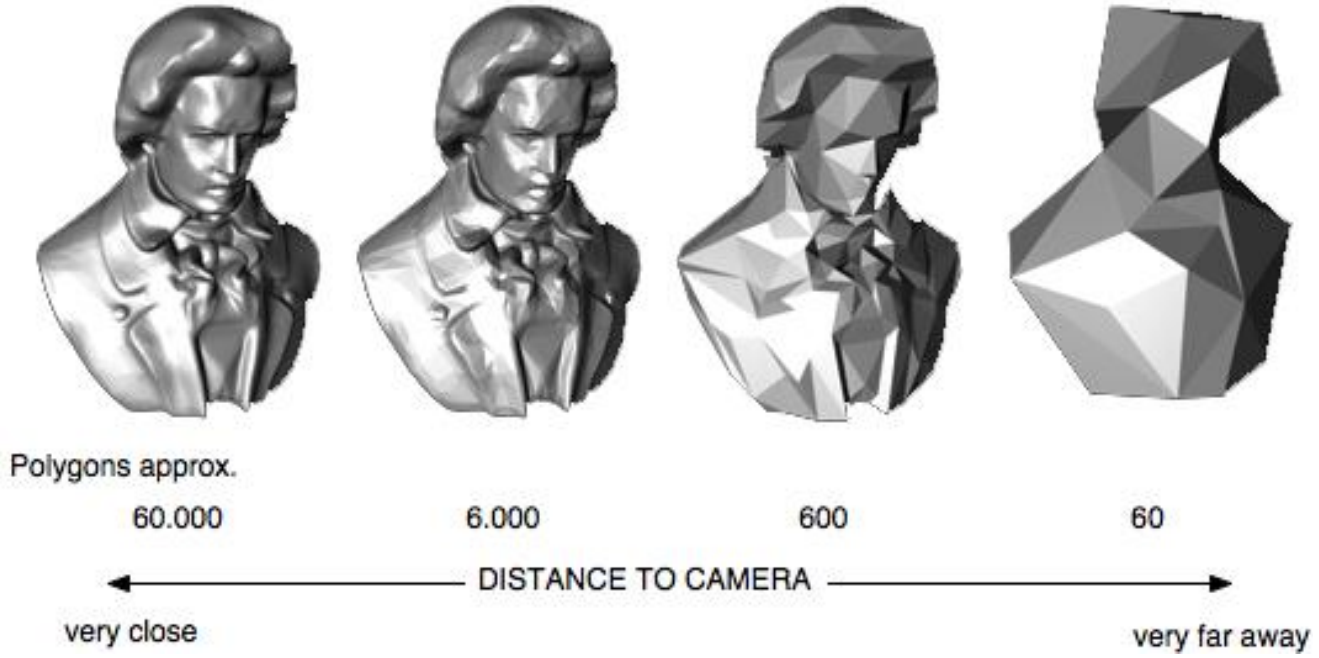


Transformation Pipeline

- Putting it all together
  - Object defined by vertices and their interconnection in object coordinates
  - Object transformed as seen in previous lecture in world coordinates
  - Object placed with the help of camera in camera coordinates
  - Projection of object in 2D projected coordinates
  - Perspective division takes place
  - Viewport is rendered in screen space

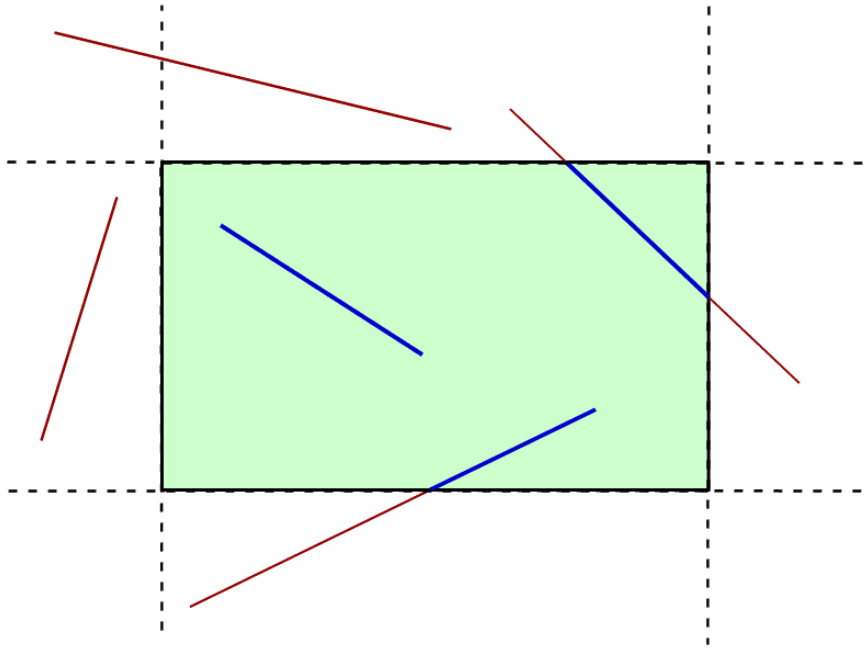
Level of Detail

- Level of Detail
  - Used to increase rendering performance by reducing the amount of drawn triangles
  - Same 3D object is reproduced with different amount of triangles
  - Choice of displayed object depending on distance between object and the camera
  - Ideally the different displayed levels of detail are not distinguishable for the user
  - Can be pre-processed or calculated dynamically
  - Mesh simplification is complex, computationally intense, and error prone
  - Variety of optimisation algorithms exist



Line Clipping

- Common algorithms to remove complete lines which are not visible in the viewing area or clipping lines which are partially visible
- Clipping is performed typically to reduce rendering time and increase performance
- Clipping operations are normally performed in 2d image space
- Variety of approaches which typically consist of the following steps
  - Determination whether the line is visible at all
  - Segmentation of the line into sub line segments which are tested again



https://en.wikipedia.org/wiki/Line_clipping#/media/File:Line2_clipping.png

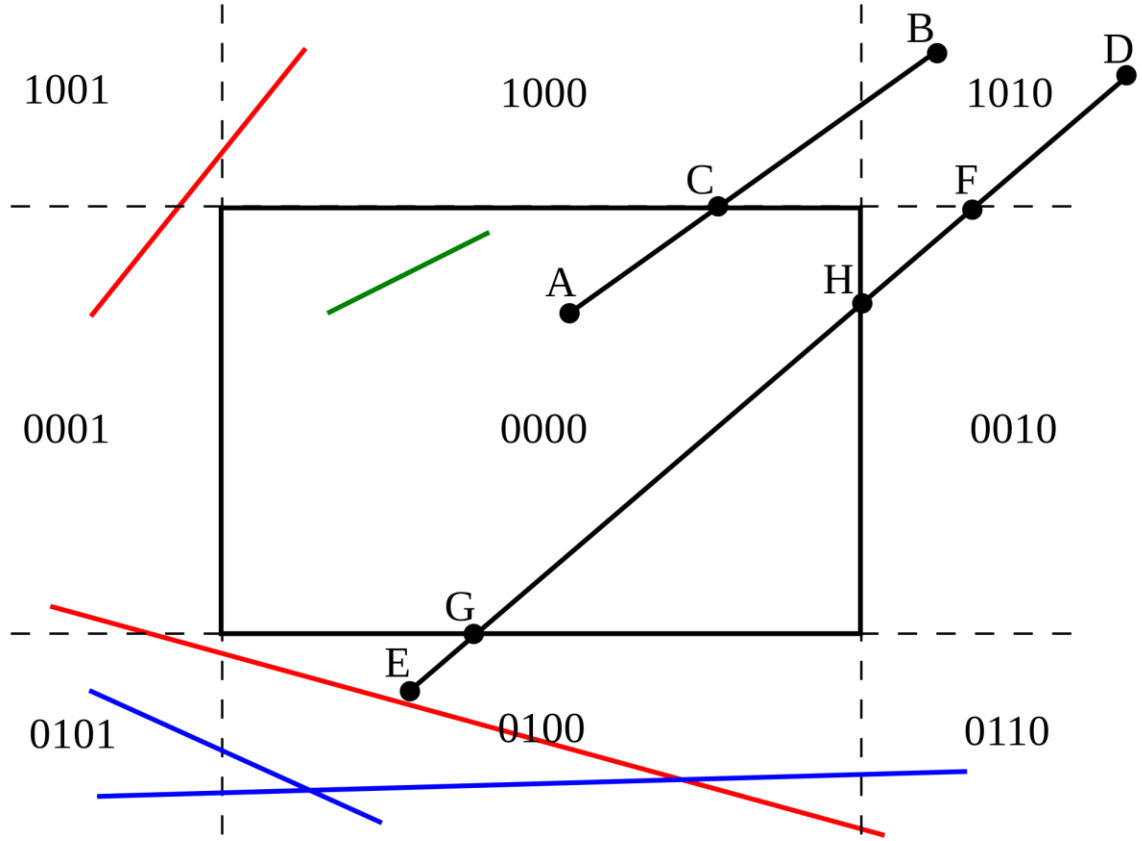
Clipping – Cohen-Sutherland Algorithm

	left	central	right
top	1001	1000	1010
central	0001	0000	0010
bottom	0101	0100	0110

- Line clipping algorithm which clips lines against a rectangular area (e.g. screen)
- Half-space code for each endpoint of a line is used composed of 2x2 bit per point
- For end points P and Q of a line
  - determine a 4 bit code for each point
    - xx01 = point is left of rectangular area; $x < x_{min}$
    - xx10 = point is right of rectangular area; $x > x_{max}$
    - 01xx = point is below (bottom) rectangular area; $y < y_{min}$
    - 10xx = point is above (top) rectangular area; $y > y_{max}$
- Code order is LRBT (Left, Right, Bottom, Top)
- Once all codes for the points are determined combine the two points of a line with a logical AND and OR operations

Clipping – Cohen-Sutherland Algorithm

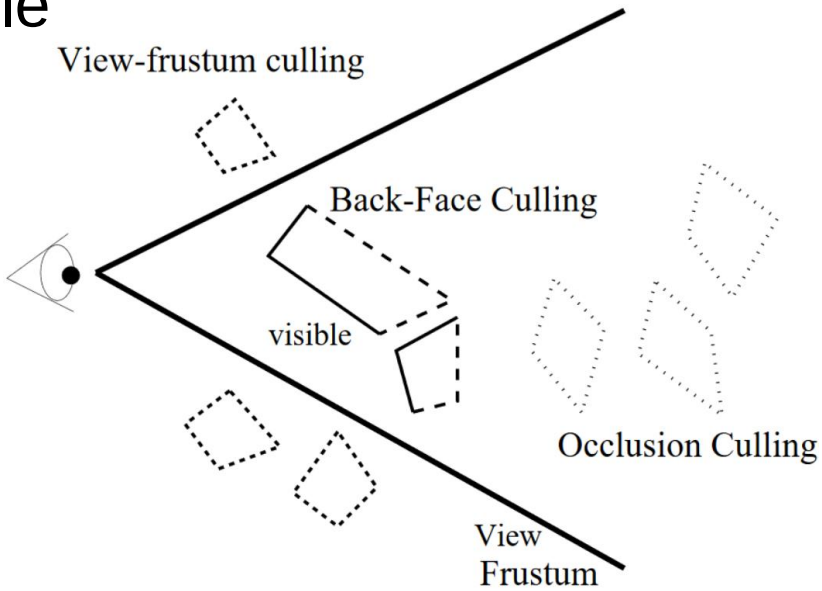
- Case distinction is performed
  - P OR Q = 0000: line is completely inside: draw as is (green line)
  - P AND Q != 0000: line lies completely on one side of rectangle: skip (blue lines)
  - In all other cases determine the segment of P (e.g. point E) and read the bits in order LRBT at the first occurrence of a 1 bit
  - Compute intersection I of the corresponding edge between P and Q (in an example E and D, resulting in G)
  - Repeat with segments until resolved



https://pl.wikipedia.org/wiki/Plik:Algorytm_Cohen_sutherland.svg

Culling Techniques

- While clipping will cut off parts during the rendering process culling is used earlier to remove whole polygons which are not visible
- Optimisation is often required or desired since calculation of colour and light can be expensive
- Only visible parts will contribute to the final image
- Most common techniques
  - Frustum culling
  - Backface culling (provides typically biggest performance increase)
  - Occlusion culling (provides typically lowest performance increase, becomes especially relevant with dense scenes with high amount of polygons)
  - Contribution culling (if objects become to small they are simply not rendered anymore)



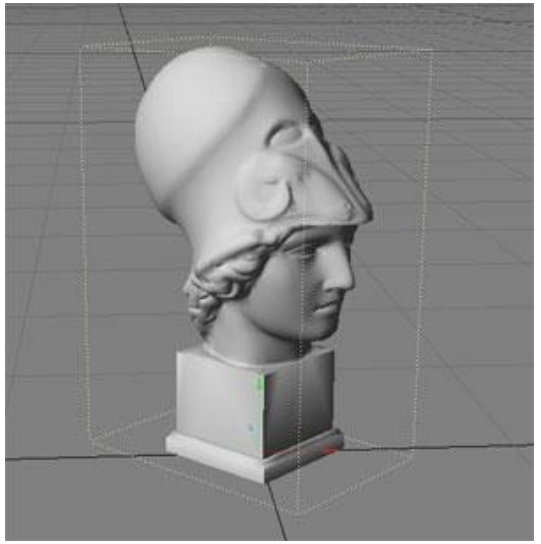
Cohen-Or, D.; Chrysanthou, Y.; Silva, C. & Durand, F. A survey of visibility for walkthrough applications IEEE Transactions on Visualization and Computer Graphics, Institute of Electrical and Electronics Engineers (IEEE), 2003, 9, 412-431

Frustum Culling

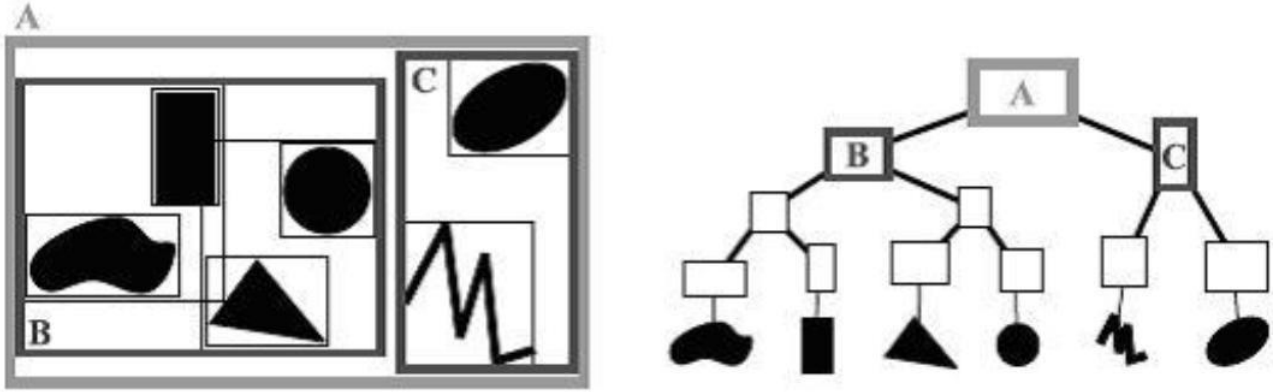
- Is active if object is outside the region of the clipping planes
- Simplified frustum culling test
  - Assuming our view vector is aligned with the z-axis
    - $p_{near} < z_{view} < p_{far}$
  - X- and Y-Axis are centric inside the viewing volume
    - $-w_{view} < x_{view} < w_{view}$
    - $-h_{view} < y_{view} < h_{view}$
- Two simple comparisons are required for each axis
- Still not ideal since test has to be performed for each triangle or primitive
- The naive frustum culling needs O(n) tests where n = number of objects, primitives or triangles
- Typically use of acceleration structures

Frustum Culling – Optimisation

- Use of bounding volumes
  - Instead of testing each triangle or primitive a bounding volume encapsulating an object based on primitives is used for testing, if the bounding volume is out of the frustum no further tests are needed
  - Bounding volumes are typically axis aligned bounding boxes (AABB)
- Bounding volume hierarchy (BVH) as we briefly heard in the scene graph section
  - If hierarchical structure of the scene is available the parent objects bounding volumes would encapsulate their children's bounding volumes
  - If parent bounding volumes is out of the frustum no child volumes would have to be tested



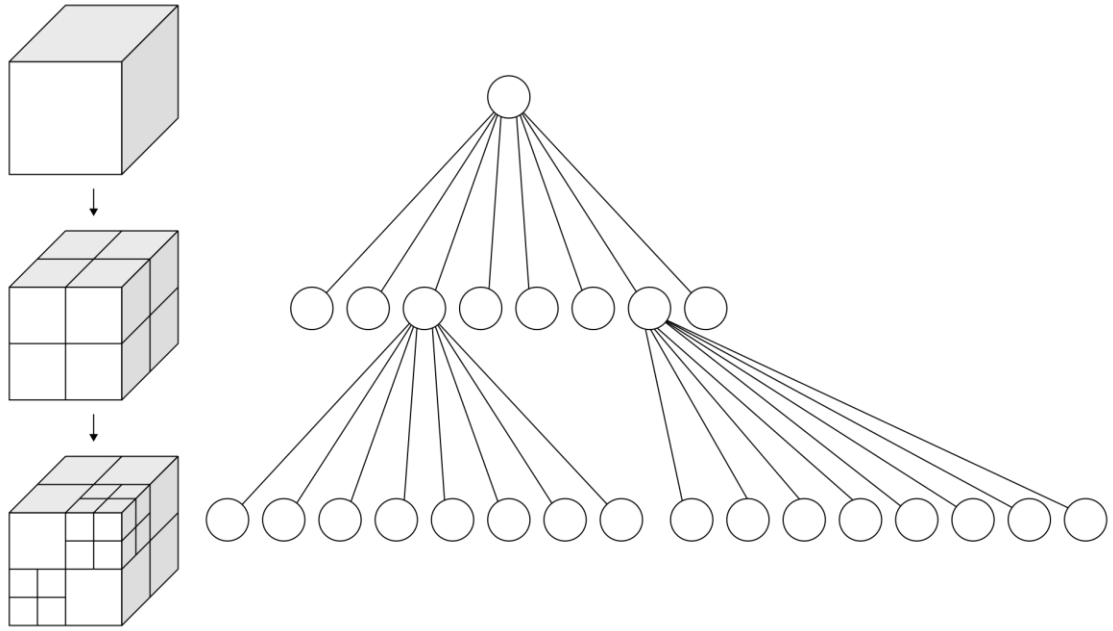
https://de.wikipedia.org/wiki/Datei:BoundingBox.jpg



https://commons.wikimedia.org/wiki/File:Example_of_bounding_volume_hierarchy.JPG

Frustum Culling – Optimisation

- Octrees
  - Can be used if no object based hierarchical structure is available (e.g. scan data)
  - Division of the entire world space into 8 cubes
    - Assign objects/primitives/triangles based on their location in world space to each cube
  - Repeat subdivision into eight cubes inside each cube
    - Repeat recursively until cube contains less than k objects



https://en.wikipedia.org/wiki/Octree#/media/File:Octree2.svg

- Instead of culling objects or bounding volumes, cull the cubes
- Needs O(log n) tests

Backface Culling

- The most simple approach of removing triangles for rendering is backface culling
- If the polygons on the back side of objects of an object form a closed surface and we perceive them from the outside we have no need for drawing the back faces of the polygons
- The polygons on the back side of objects face backwards

https://upload.wikimedia.org/wikipedia/commons/0/0c/Back_face_culling_skull_example.png

- Depending on the dataset quite often source of error

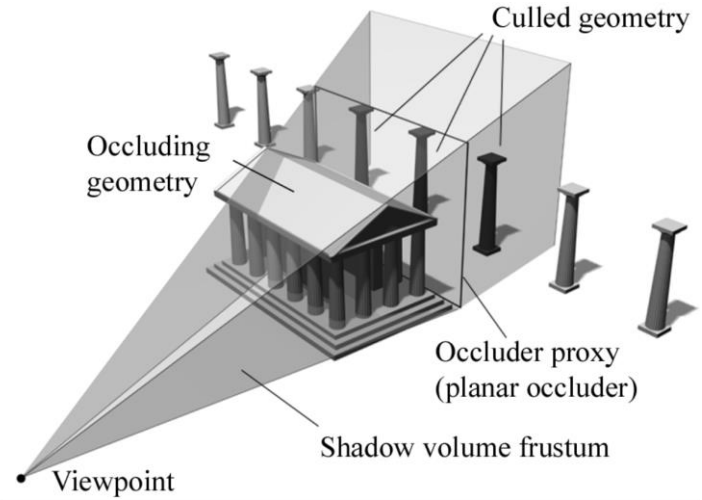
Backface Culling

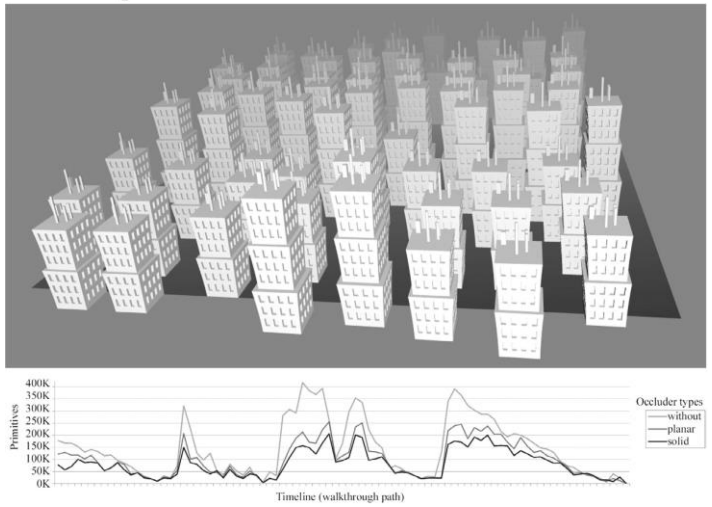
- We use the normal vector of the polygon to check for orientation
  - Typically normal vectors are stored in face mesh structure
  - If not we can calculate the normal vector as cross product of 2 triangle edges
- The normal faces backwards if angle between the optical axis and the normal vector is > 90° (i.e. scalar product is > 0)

- Backface Culling can be extremely efficient
  - In scenes with many solid objects the amount of polygons to be rendered can often be reduced by around 50%
  - What type of scenes do no not profit that much from backface culling?

Occlusion Culling

- Objects that are completely hidden behind other objects in the scene do not have to be drawn
- Has to be considered globally, while backface culling and frustum culling can be considered locally
- Planar Occluders
  - Uses projection of objects on viewing plane blocking area for depth tests later on
  - Can be implemented by scene geometry or using separate occlusion proxies
- Shadow Volumes
  - Creation of a shadow volume between the viewpoint and the far clipping plane
  - More on that in texture and buffer part of the lecture





Papaioannou, G.; Gaitatzes, A. & Christopoulos, D. Efficient Occlusion Culling using Solid Occluders *WSCG'2006,* 2006

Bibliography

- Carlbom, I. & Paciorek, J. Planar Geometric Projections and Viewing Transformations ACM Computing Surveys, Association for Computing Machinery (ACM), 1978, 10, 465-50
- James, O.; von Tunzelmann, E.; Franklin, P. & Thorne, K. S. Gravitational lensing by spinning black holes in astrophysics, and in the movie Interstellar Classical and Quantum Gravity, IOP Publishing, 2015, 32, 065001
- Sutherland, I. E. & Hodgman, G. W. Reentrant polygon clipping Communications of the ACM, Association for Computing Machinery (ACM), 1974, 17, 32-42
- Cohen-Or, D.; Chrysanthou, Y.; Silva, C. & Durand, F. A survey of visibility for walkthrough applications IEEE Transactions on Visualization and Computer Graphics, Institute of Electrical and Electronics Engineers (IEEE), 2003, 9, 412-431
- Papaioannou, G.; Gaitatzes, A. & Christopoulos, D. Efficient Occlusion Culling using Solid Occluders WSCG'2006, 2006
- Fuchs, H.; Kedem, Z. M. & Naylor, B. F. On visible surface generation by a priori tree structures ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1980, 14, 124-133

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Computer Graphics

Illumination, Reflectance and Shading Models

FH-Prof. Dr. techn. Dipl.-Inf. (FH) Christoph Anthes, MSc

Professor of Augmented and Virtual Reality

Overview

- Light Sources
  - Types of Light Sources
- Illumination or Reflectance Models
  - Lambert, Diffuse, Phong
- Advanced Reflectance Models
  - BRDF
  - Subsurface Scattering
- Shading Models
  - Flat, Gouraud, Phong
- Transparencies

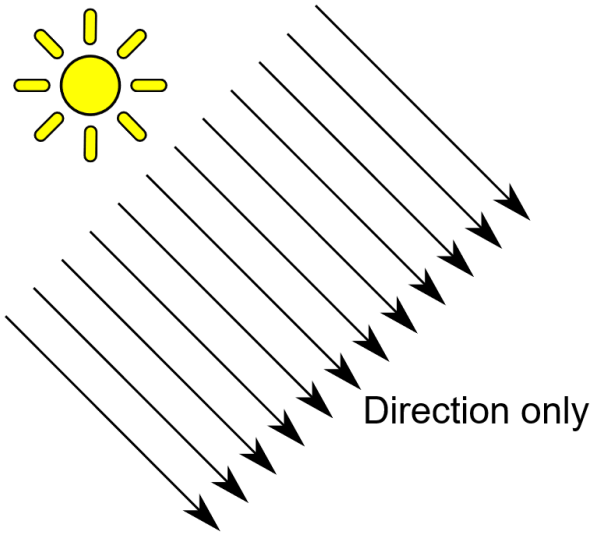
Light and Light Sources

- The intensity of the light received from a surface in terms of colour (based on the chosen colour model) depends on
  - The light sources in the environment
  - The structure and other properties of the material hit by the light
  - The description on how the light is reflected (illumination or reflectance model)
  - The description on how the surface is shaded (shading model)
- Light sources
  - Light sources are used to simulate physical lights in the scene
  - Depending on the chosen model this can be unrealistic and computationally cheap or realistic and computationally expensive
  - The have a given intensity and colour
  - Their position and type as well as, depending on the type of light source, orientation is important

Light Sources

- Ambient Light (indirect)
  - A light source which is not directed
  - Can be considered as environmental light
  - Light is evenly distributed in all directions
  - Creates low contrast images

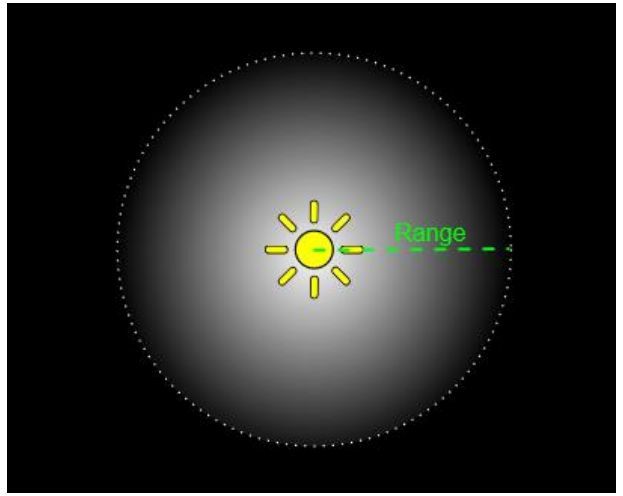
- Directional Light (direct)
  - Light is emitted from a far distance, so that the rays are considered parallel
  - The intensity of the light is independent of the distance between objects and the light source
  - The range of a directional light is typically unlimited. (e.g. sun)



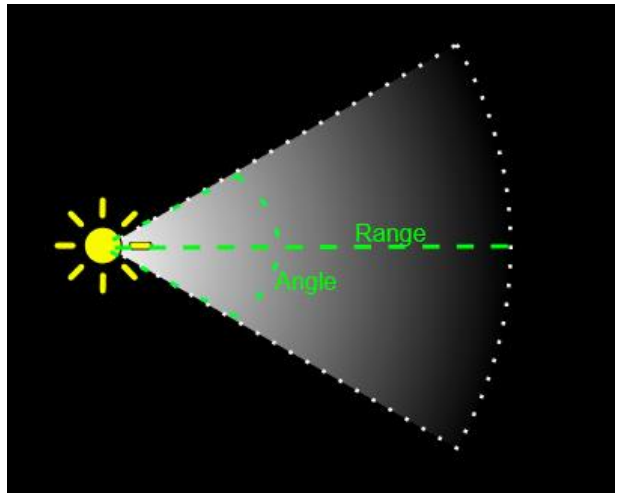
https://docs.unity3d.com/Manual/Lighting.html

Light Sources

- Point Light (direct)
  - Light source with a single point of origin illuminating equally in all directions
  - Depending of the distance from the illuminated objects to the light source the intensity of the light is determined
  - Range is typically limited (e.g. light bulb)



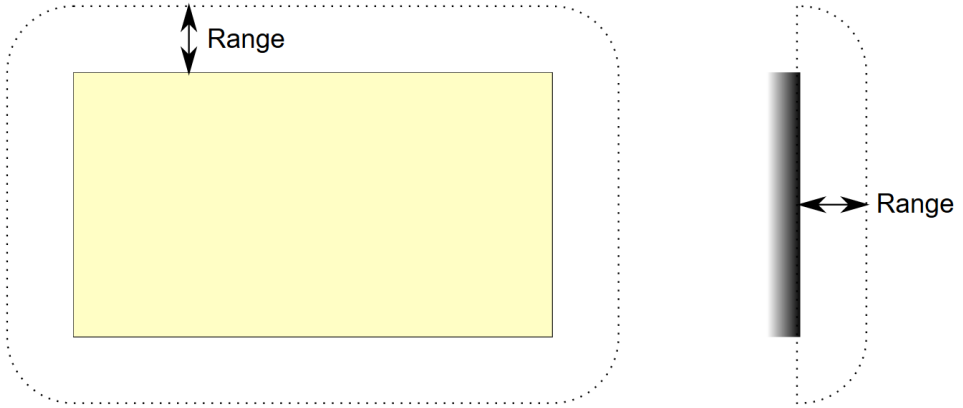
- Spot Light (direct)
  - Is a point light emitting light with a given opening angle
  - A direction for the central light ray is required
  - The range is as with a regular point light limited
  - The intensity depends as well on the distance between light source and objects (e.g. flashlight)



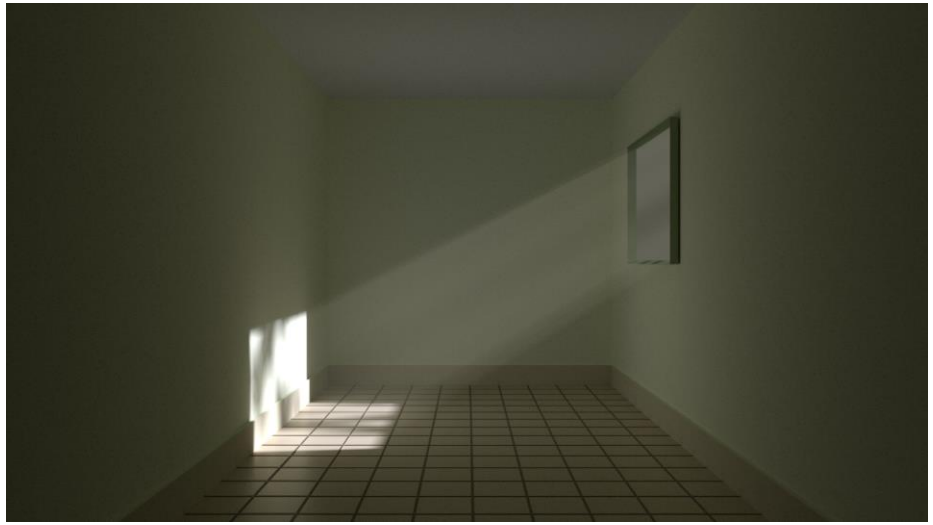
https://docs.unity3d.com/Manual/Lighting.html

Light Sources

- Area Light
  - Is described by an object geometry and the light intensity
  - Typically implemented by multiple light sources
  - Light is emitted in one direction from the shape
  - Computationally quite intense
  - Attenuation restricts the range of the light source
- Volumetric Light
  - Illuminates a certain region of an arbitrary predefined shape (e.g. cube)
  - Implemented with light space shadow maps and the depth buffer (more on that in buffer section of the course)



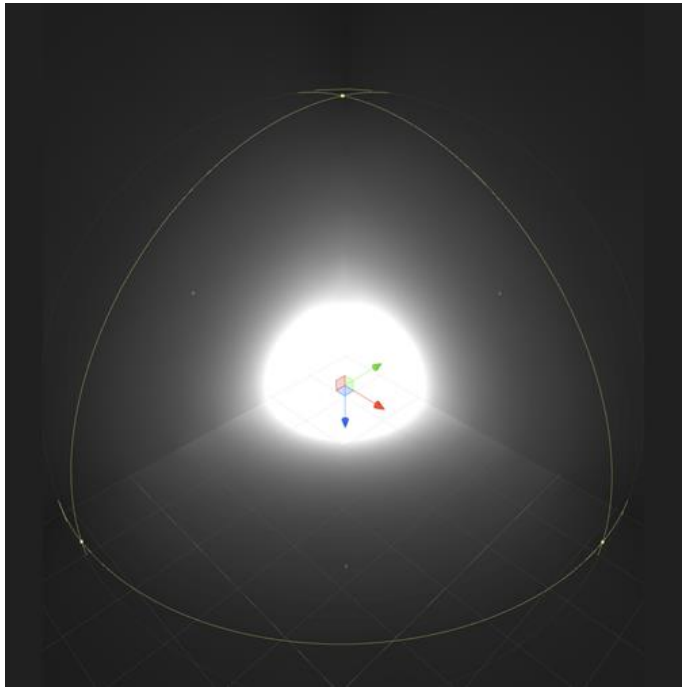
https://docs.unity3d.com/Manual/Lighting.html

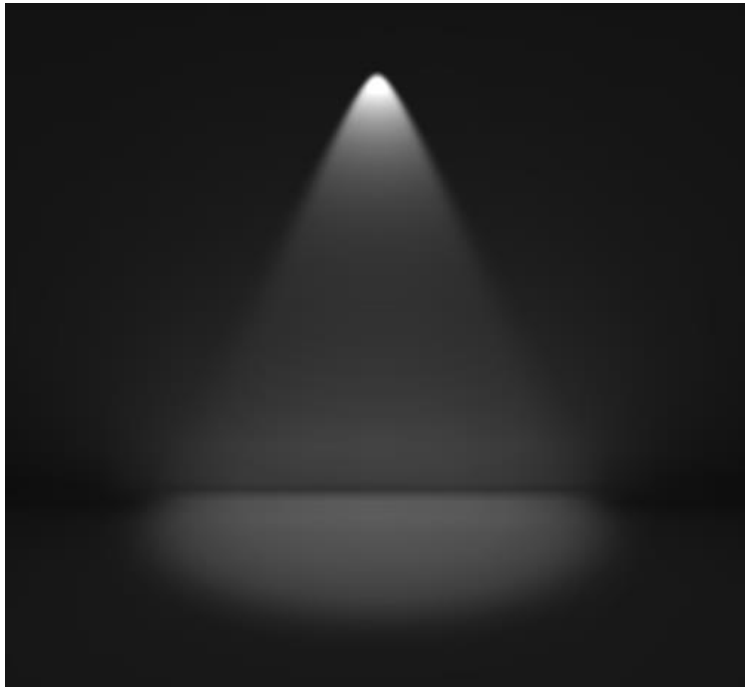


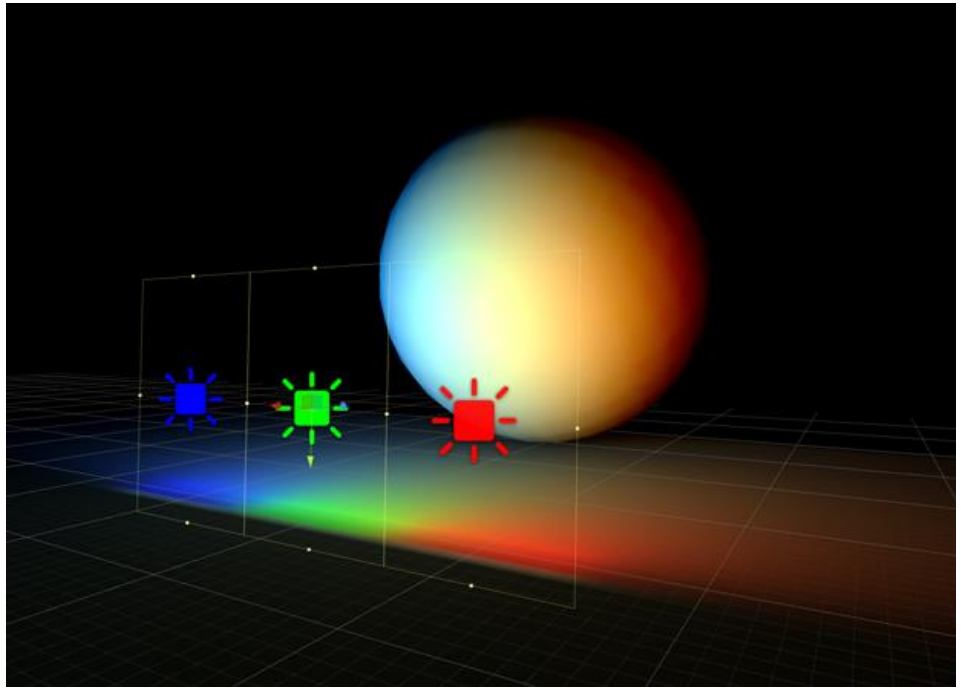
https://en.wikipedia.org/wiki/Volumetric_lighting#/media/File:Environment_lighting.png

Examples for Light Sources

- Examples of common light sources in Unity (first two available in OpenGL)







https://docs.unity3d.com/Manual/Lighting.html

Point Light

Spot Light

Area Light

The Rendering Equation

- Integral equation formulating in a generalised way for light transport in a scene
- Foundation for many rendering techniques in CG although published in 1986

$$L_o(x, \omega_o, \lambda, t) = L_e(x, \omega_o, \lambda, t) + \int_\Omega f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_o, \lambda, t) (\omega_i \cdot n) d\omega_i$$

L_o - the outgoing light

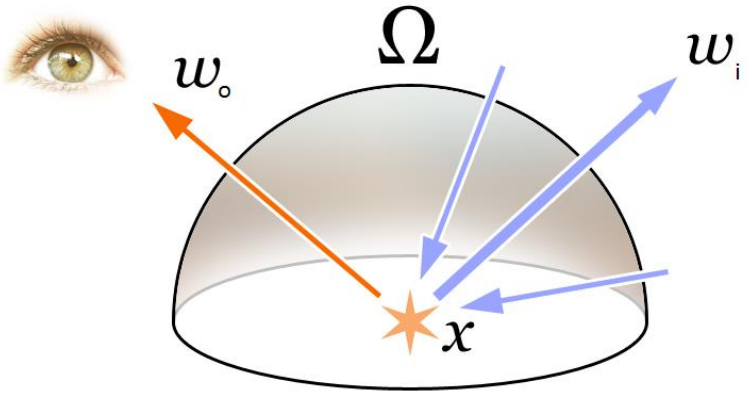
L_e - the emitted light

– Reflectance function

L_i - incoming light

– the angle of incoming light

- Not all aspects of reflection are considered or can be modelled by the rendering equation (e.g. subsurface scattering, transmission)



https://upload.wikimedia.org/wikipedia/commons/d/d1/Rendering_eq.png

Illumination or Reflectance Models

- Illumination or reflectance models describe how the light is reflected on the surface of an object
- The light ray emitted from **a light source** hits the **surface** of an object in a given **angle**, depending on the **surface properties** of the object it is determined by the **reflectance model** how the light is reflected
- This reflection contributes significantly to colour of the object in the final image
- Considering the rendering equation the reflectance models describe the term of the reflectance function

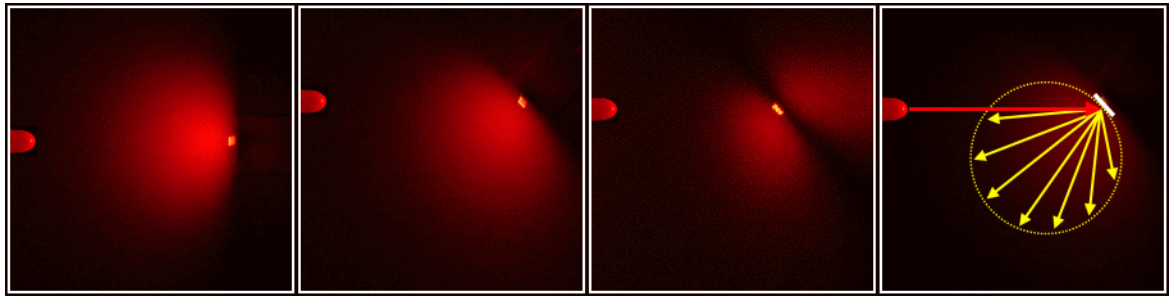
Illumination or Reflectance Models

- Most simple lighting model
  - The most simple model would be ignoring light sources and setting a constant illumination for the surface based on a defined value
  - Computationally extremely cheap

- Ambient Lighting
  - Equally balanced background light caused for example by a multitude of reflections of the surroundings
  - Every surface, independent from its orientation, receives the same amount of ambient light
  - All surfaces of an object with the same material will reflect in the same way in the same defined material colour
  - Computationally cheap

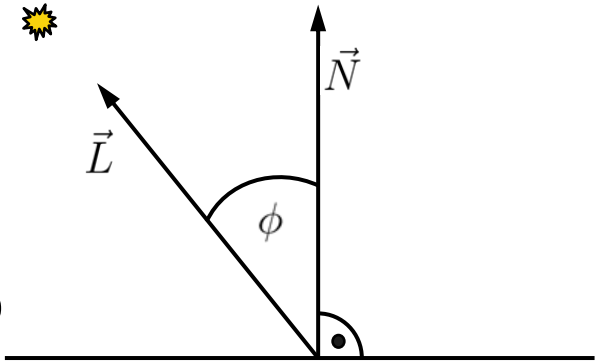
$$I_{\text{ambient}} = k_a I_a$$

Illumination or Reflectance Models



https://upload.wikimedia.org/wikipedia/commons/2/2f/Lambertcosrp.png

- Diffuse Reflection
  - With diffuse reflection the light is evenly reflected in all directions with same intensity
  - It appears on matte and dull surfaces (e.g. chalk)
- Lambertian Cosine Model
  - The intensity of the light is proportional to the scalar product of the surface normal $\vec{N}$ and the vector $\vec{L}$ pointing to the light source
  - The more direct (angle between light source and surface is closer to perpendicular) the light source illuminates the surface, the more light is reflected (cos(0)=1; cos(90)=0)
  - If Phi is between 0° and 90° the surface is hit directly by the light (for negative values of cos Phi the light source is behind the surface)
  - The intensity I_{diffuse} is calculated for an ideal diffuse surface $I_{\text{diffuse}} = k_d I_d \cos\phi$



Illumination or Reflectance Models

- When $\vec{N}$ and $\vec{L}$ are normalized we can replace $\cos\phi$ with the scalar product of the two vectors $(\vec{L}\cdot\vec{N})$ resulting in $I_{\text{diffuse}} = k_d I_d(\vec{L}\cdot\vec{N})$

- Attenuation
  - With many models attenuation is not considered, for example if two surfaces are far away (relative to each other) on the depth axis but close by on the other axes they will result in the same reflected colour
  - Based on the distance of the surface to the light source the intensity of the reflected light degrades, thus an additional attenuation factor can be added

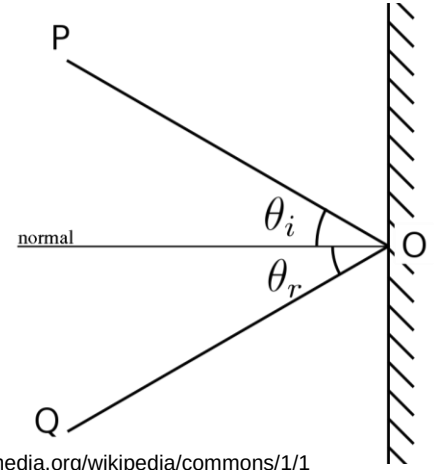
$$I = I_a k_a + f_{\text{att}} I_d k_d(\vec{N}\cdot\vec{L})$$

  - When we take into account that energy falls of at the inverse square of the distance it travels d_L we can approximate $f_{\text{att}} = \frac{1}{d_L^2}$

Illumination or Reflectance Models

- Specular Reflection
  - This type of reflection can be observed on shiny surfaces (e.g. mirror, lake, pool ball, apple)
  - An incident light ray hits a surface at a given angle, the light is then reflected from the surface at the exact same angle mirrored along the normal vector
  - If it is not an ideal mirror a highlight is caused by specular reflection
  - The specular highlight has the colour of the incident light (e.g. white)
  - The reflection is also affected by positional changes of the observer or the camera

https://upload.wikimedia.org/wikipedia/commons/e/ef/Mount_Hood_reflected_in_Mirror_Lake%2C_Oregon.jpg



https://upload.wikimedia.org/wikipedia/commons/1/10/Reflection_angles.svg

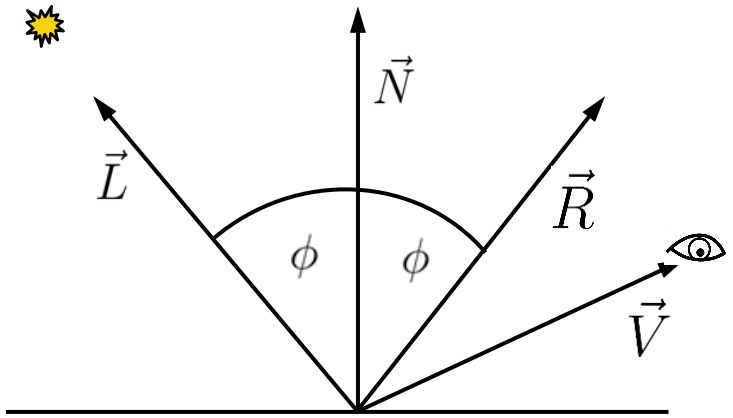
Illumination or Reflectance Models

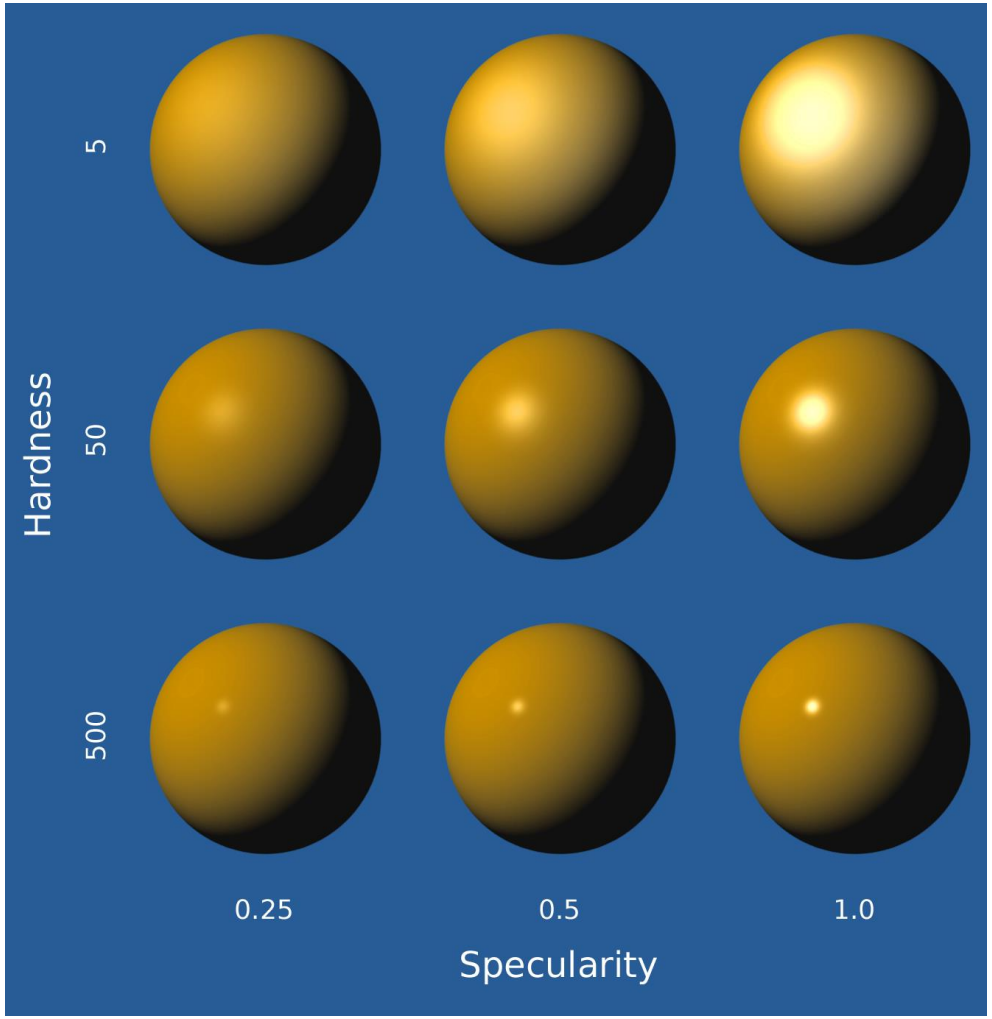
- Specular Reflection
  - For calculating a specular reflection we will need the vector $\vec{R}$ which is mirrored $\vec{L}$ along the normal $\vec{N}$

$$\vec{R} = 2(\vec{L} \cdot \vec{N})\vec{N} - \vec{L}$$

  - Additionally we require a shininess constant α describing the specularity of the object

$$I_{\text{specular}} = I_s k_s (\vec{R} \cdot \vec{V})^{\alpha}$$





Illumination or Reflectance Models

- Combination of the reflectance models we have seen so far can be integrated into one model
- If multiple light sources are to be considered the individual values per light source are calculated and afterwards summed up to generate the resulting colour

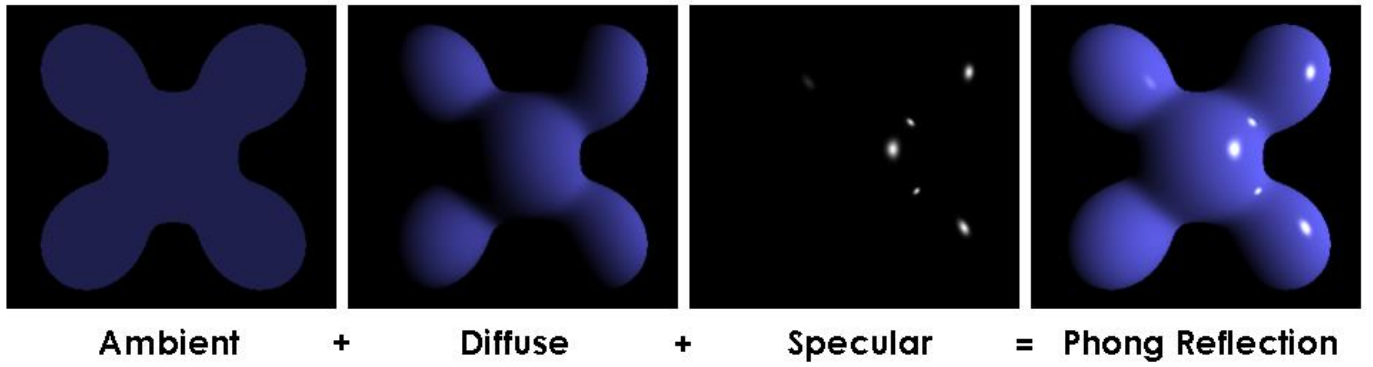
$$I_{\text{phong}} = \underline{I_a k_a} + \sum_{m \in \text{lights}} (\underline{k_d(\vec{L_m} \cdot \vec{N}) I_{m,d}} + \underline{k_s(\vec{R_m} \cdot \vec{V})^{\alpha} I_{m,s}})$$

  - Ambient component (once in general)
  - Diffuse component (for each light source)
  - Specular component (for each light source)

Illumination or Reflectance Models

- Phong Reflectance Model
  - Splits reflectance into 3 different components
  - Can be individually calculated

$$I_{\text{phong}} = I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}}$$



https://upload.wikimedia.org/wikipedia/commons/6/6b/Phong_components_version_4.png

- Phong - Ambient Component

$$I_{\text{ambient}} = k_a I_a$$

with

k_a - ambient material constant

I_a - ambient light hitting the surface

http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter05.html

  - Results in a single ambient colour depending only on light intensity and ambient reflectance

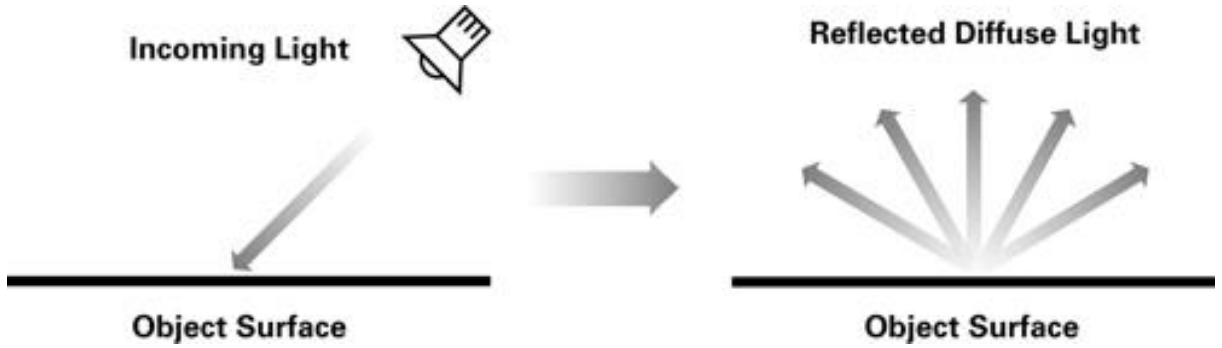
Illumination or Reflectance Models

- Phong - Diffuse Component
  - Calculated by Lamberts cosine law

$$I_{\text{diffuse}} = k_d I_d \cos\phi$$

$$I_{\text{diffuse}} = k_d I_d (\vec{L} \cdot \vec{N})$$

k_d - the diffuse reflection constant is determined empirically



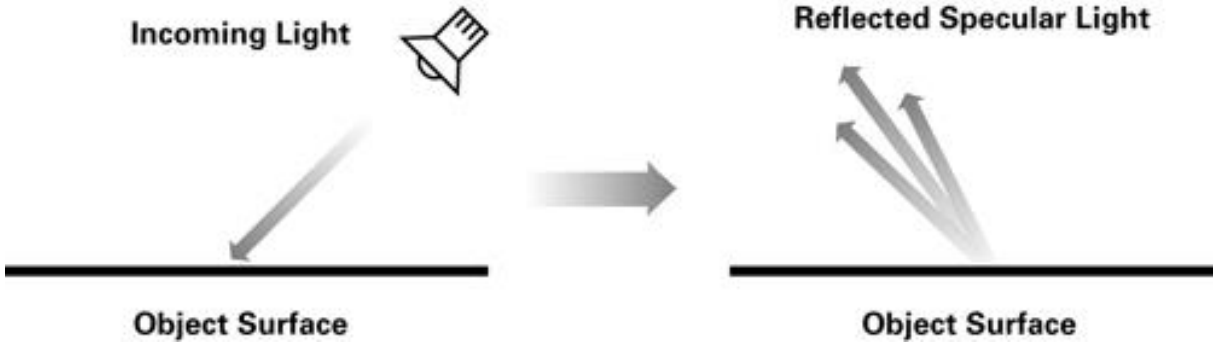
http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter05.html

- Phong - Specular Component

$$\vec{R} = 2(\vec{L} \cdot \vec{N})\vec{N} - \vec{L}$$

$$I_{\text{specular}} = I_s k_s (\vec{R} \cdot \vec{V})^{\alpha}$$

k_s - the specular reflection constant is determined empirically



http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter05.html

Illumination or Reflectance Models

- Parameters of Phong

$$I_{\text{phong}} = I_a k_a + \sum_{m \in \text{lights}} (k_d(\vec{L_m} \cdot \vec{N}) I_{m,d} + k_s(\vec{R_m} \cdot \vec{V})^{\alpha} I_{m,s})$$

k_s
- if k_s = 0 the material will be perfectly matte
- use k_s > 0 to make the material more glossy

k_a
- chose k_a > 0 to avoid harsh shadows
- keep k_a small to avoid a glowing effect of objects

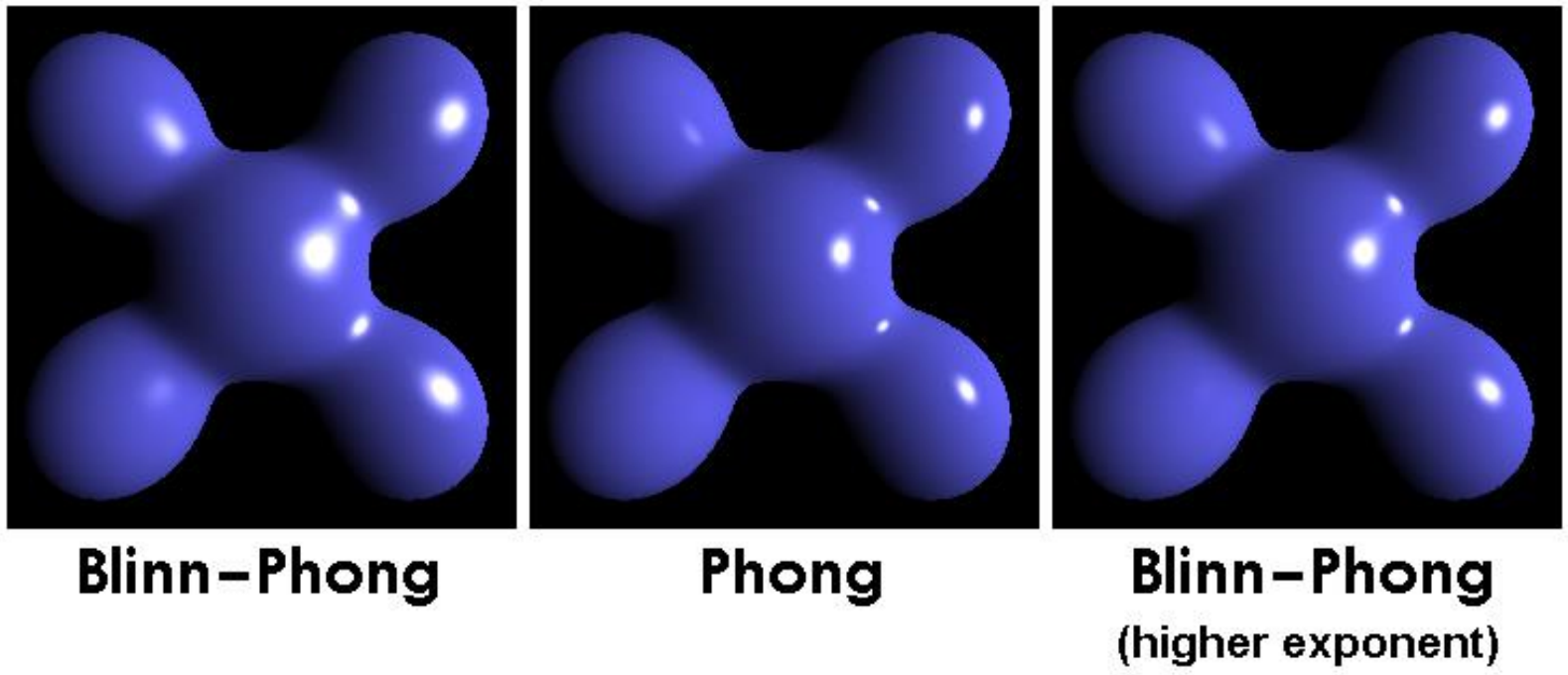
α
- adjust the size of specular highlights with alpha

Illumination or Reflectance Models

- Blinn-Phong
  - Very similar to Phong reflectance but improved in terms of rendering performance
  - Problem with Phong reflectance is constant evaluation of $(\vec{R} \cdot \vec{V})$
  - Can be approximated by using the halfway vector H

$$H = \frac{L + V}{\|L + V\|}$$

  - Can improve performance if light sources are static
  - Similar appearance in rendering but faster computation

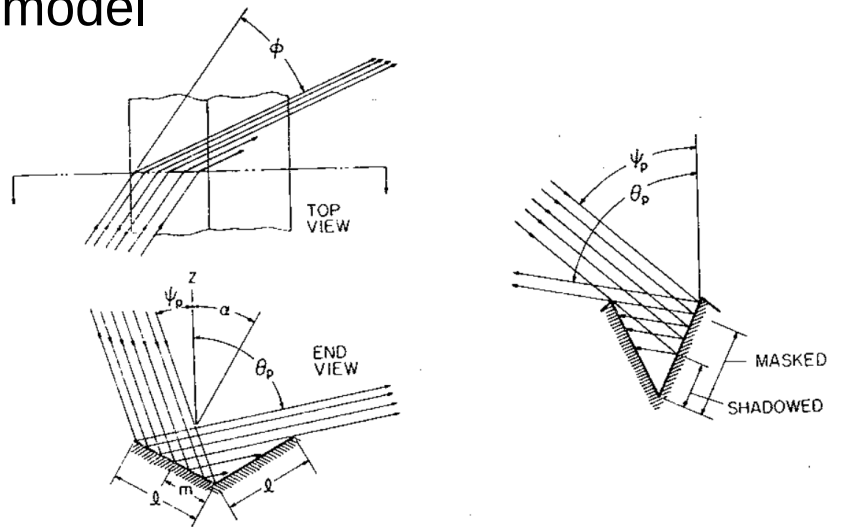


https://de.wikipedia.org/wiki/Blinn-Beleuchtungsmodell#/media/File:Blinn_phong_comparison.png

Advanced Reflectance Models

- Alternatives exist for example
  - Oren–Nayar
    - Extend Lambertian reflection by adding a roughness to the reflection
    - It assumes the surface to be composed of long symmetric V-cavities consisting of two facets
    - Each of these facets follows a Lambertian reflectance model
  - Torrance-Sparrow
    - Extends Lambertian reflection by generating rough opaque specular surfaces (glossy surfaces)
    - As with Oren-Nayar the surface consists of facets
    - Each of these facets follows a specular reflectance model

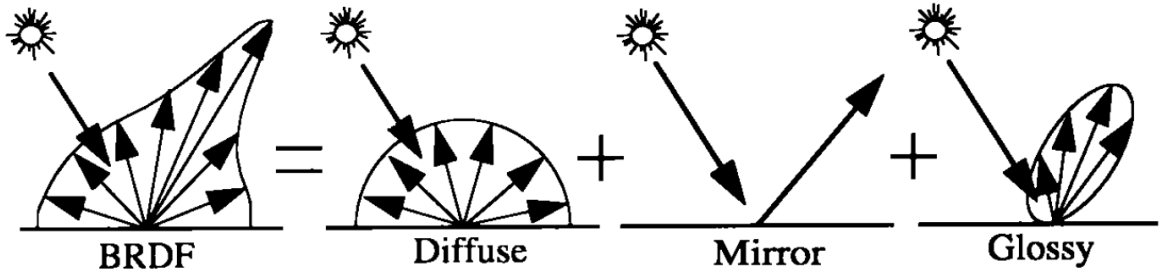
https://en.wikipedia.org/wiki/Oren%E2%80%93Nayar_reflectance_model#/media/File:Oren-nayar-vase2.jpg



Torrance, K. E. & Sparrow, E. M. Theory for Off-Specular Reflection From Roughened Surfaces∗ Journal of the Optical Society of America, The Optical Society, 1967, 57, 1105

Advanced Reflectance Models

- Bidirectional Reflectance Distribution Function (BRDF)
  - Based on physical model rather than geometrical model
  - Calculates for each hitting light ray the ratio of reflected **radiance** exiting the surface hitpoint to the **irradiance** incident on the surface



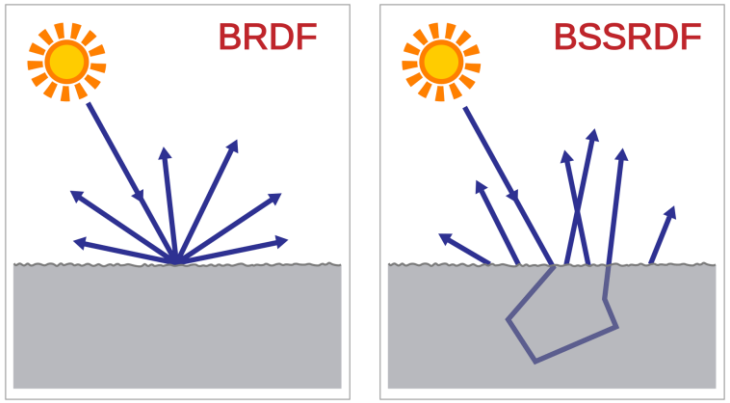
Cohen, M. F. Radiosity and Realistic Image Synthesis Morgan Kaufmann, 2016

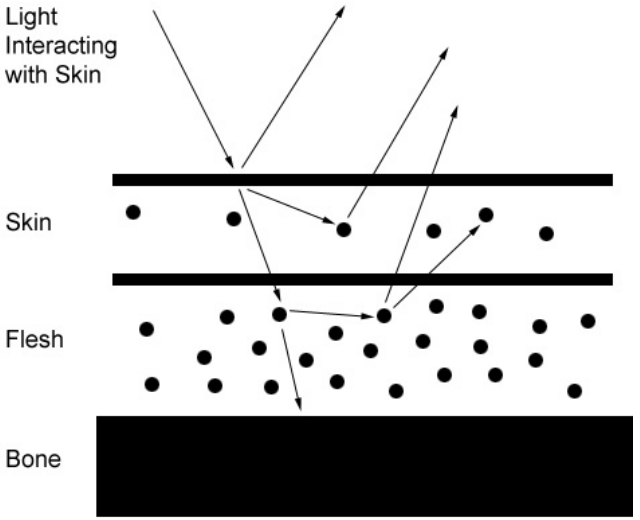
  - **Irradiance** at point of a surface is the density of radiant flux (power) per unit of surface area
  - **Radiance** is the amount of radiant flux emitted/transmitted/received per unit projected area, per unit solid angle. It involves a direction!

  - In practice the concept combines diffuse, mirror and glossy reflection

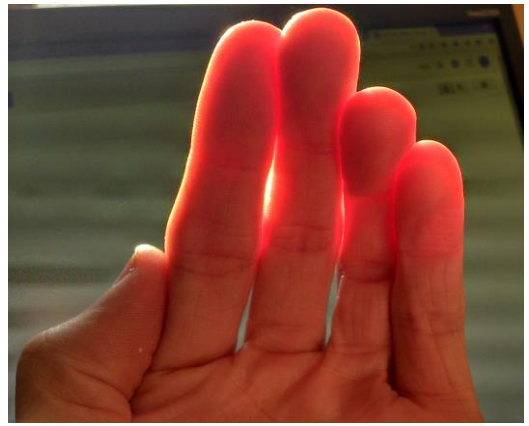
Advanced Reflectance Models

- Subsurface Scattering
  - Some materials have complex reflectance models (e.g. marble, skin, wax, hair, milk and other fluids) - applies to most non-metals
  - Partial transmission of light through a surface with partial back reflection of the light rays has to be modelled
  - Back reflection of light rays in the surface causes again diffuse reflection on the surface itself
  - One approach is to model the reflectance of multiple layers in multiple textures and use them for calculation
  - Can be modelled as an extension of BRDFs





Brown University, cs123 INTRODUCTION TO COMPUTER GRAPHICS, 2015 course notes



https://upload.wikimedia.org/wikipedia/commons/5/50/Skin_Subsurface_Scattering.jpg

Advanced Reflectance Models - Example

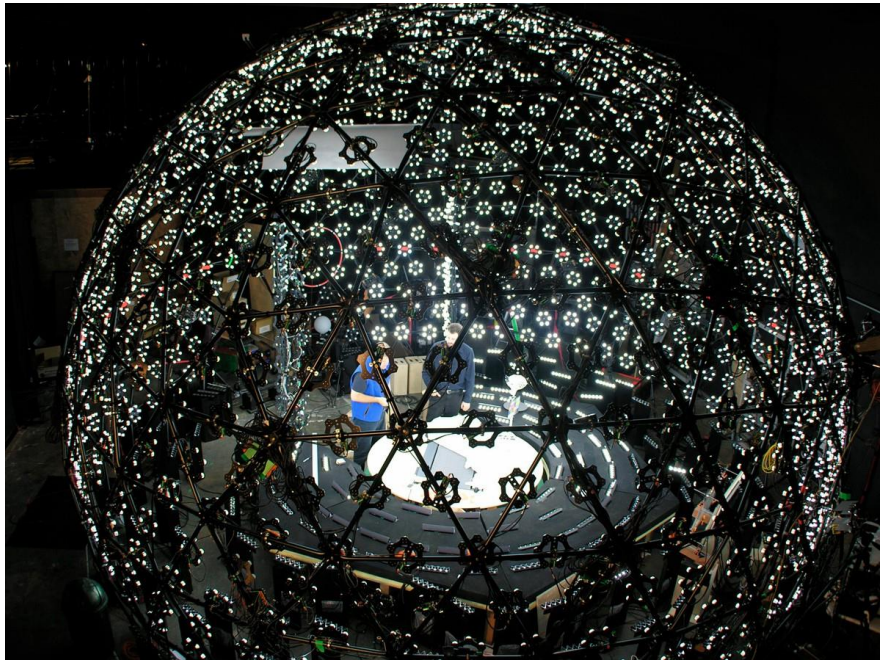
Jorge Jiminez – Separable Subsurface Scattering https://www.youtube.com/watch?v=P0Tkr4HaIVk

Advanced Reflectance Models

- Subsurface Scattering
  - Advanced methods try to capture subsurface scattering and generate these maps

https://upload.wikimedia.org/wikipedia/commons/7/7e/ShellOpticalDescattering.png

- Light Stage (http://gl.ict.usc.edu/lightstages/)
  - Multiple iterations of the system exist
  - Illumination of the user inside the light stage
  - Recording of the reflectance by multiple cameras
  - Application mainly Hollywood movies (e.g. Spiderman 2, Benjamin Button, Avatar, …), but also cultural heritage and documentation (e.g. Barack Obama)
  - Realistic rendering of actors with arbitrary lighting conditions possible



https://www.fxguide.com/featured/paul-debevec-and-ict-an-fxphd-bkd/

Shading Models

- The calculation of the colour of an object is determined by a shading model which uses the calculations from the previously introduced reflectance models
- Shading can be applied on a per surface basis (flat or constant shading) or it can be applied on a per pixel basis (Gouraud shading, Phong shading)

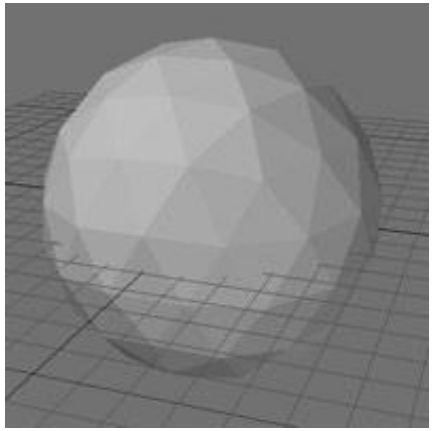
- A shader is a program that was originally used for implementing the shading mechanisms
- Shaders are currently used for a variety of applications beyond shading mechanisms (e.g. General Purpose GPU (GPGPU) programming)
- Shading languages are rather low-level and optimised for graphics cards where executed

```
varying vec3 N;
varying vec3 v;

void main (void)
{
   vec3 L = normalize(gl_LightSource[0].position.xyz - v);
   vec3 E = normalize(-v); // we are in Eye Coordinates, so EyePos is (0,0,0)
   vec3 R = normalize(-reflect(L,N));

   //calculate Ambient Term:
   vec4 Iamb = gl_FrontLightProduct[0].ambient;

   //calculate Diffuse Term:
   vec4 Idiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0.0);
   Idiff = clamp(Idiff, 0.0, 1.0);

   // calculate Specular Term:
   vec4 Ispec = gl_FrontLightProduct[0].specular
                * pow(max(dot(R,E),0.0),0.3*gl_FrontMaterial.shininess);
   Ispec = clamp(Ispec, 0.0, 1.0);

   // write Total Color:
   gl_FragColor = gl_FrontLightModelProduct.sceneColor + Iamb + Idiff + Ispec;
}
```

https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/lighting.php

Shading Models

- Flat or Constant Shading
  - The most simple and fast shading model is the flat shading
  - A whole triangle is assigned a single colour value
  - The colour of the triangle can be determined in multiple ways
    - The colour of the first vertex with its normal can be used
    - The colour of the centroid can be used with help of the face normal
    - The colour of the averaged vertices of the triangle can be used
  - Between the individual triangles the edges become visible, leading to clearly visible distinct surfaces (Mach-band effect) exaggerating contrast
  - Transparency, shadows and highlights are normally ignored with flat shading
  - Can find use in applications, where a high amount of triangles needs to be rendered simultaneously and the display quality is not as important (e.g. industrial CAD files)
  - Can also be used for artistic purposed to generate a retro look (e.g. digital games)

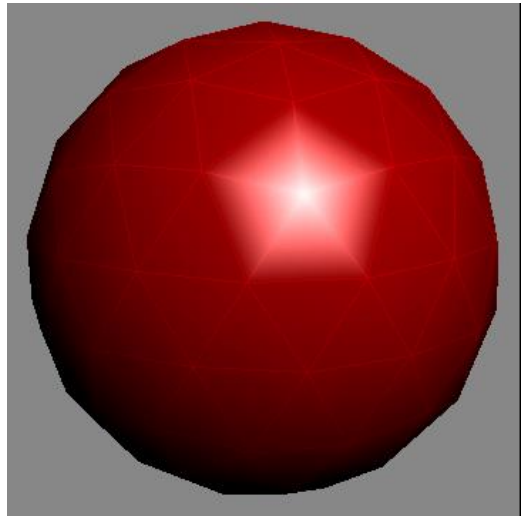


https://upload.wikimedia.org/wikipedia/commons/c/c2/Flat_shading.jpg

https://en.wikipedia.org/wiki/Mach_bands#/media/File:Mach_bands_-_animation.gif

Shading Models

- Gouraud Shading
  - Calculates the individual colours of the vertices
  - The colour of the triangle is determined by interpolation of the vertex colours
- Basic Algorithm
  - Calculate the vertex normal vectors if not already stored or available (this is achieved by averaging the normal of all adjacent face normals)
  - Calculate the colour intensity of each vertex normal depending on the current local illumination model
  - Calculate the colour intensities of the edges by linear interpolation of the vertex normal
  - Calculate the colour intensities of the inside of the polygon by using linear interpolation of the edges

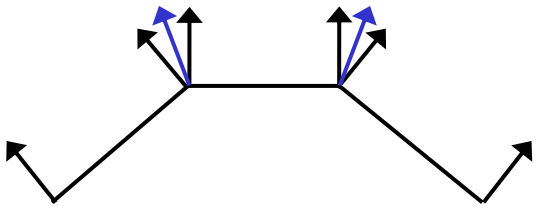


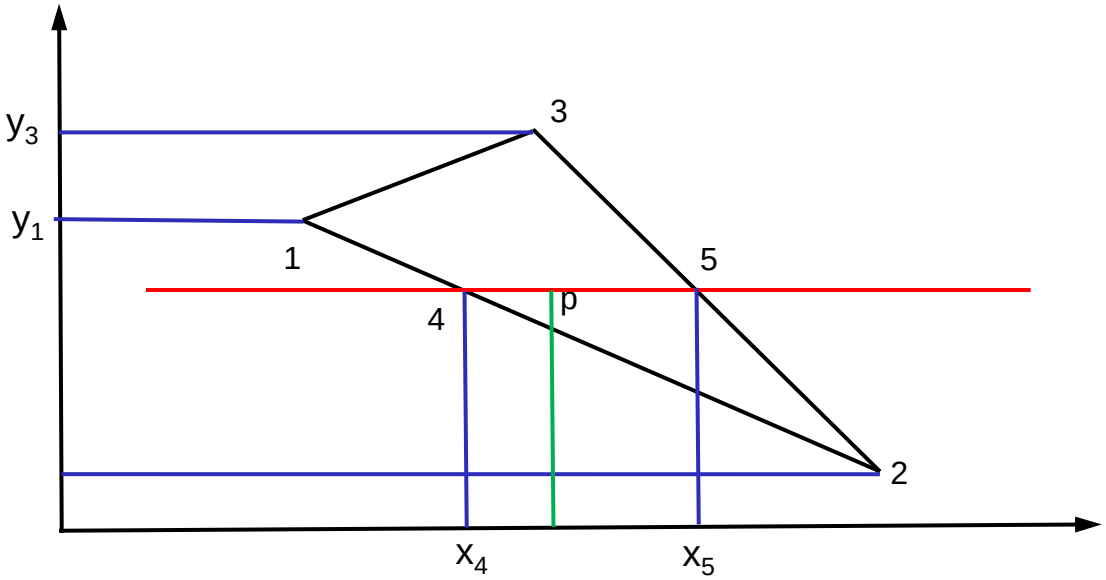
https://en.wikipedia.org/wiki/Gouraud_shading#/media/File:Gouraud_low_anim.gif

https://6images.cgames.de/images/gamestar/279/star-wars-tie-fighter_2267765.jpg

Shading Models

- Gouraud Shading
  - Face calculation is achieved by using scan line algorithm
  - Interpolation of P_4 and P_5 based on their edge values (P_1 and P_2 respectively P_3 and P_2)
  - Interpolation of P based on the previously calculated P_4 and P_5





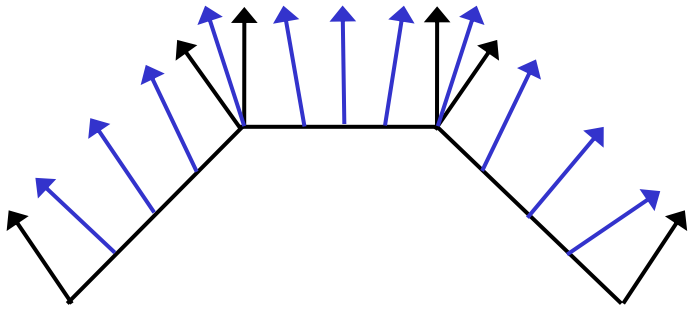
$$P_4 = \frac{y_4 - y_2}{y_1 - y_2} P_1 + \frac{y_1 - y_4}{y_1 - y_2} P_2$$

$$P_5 = \frac{y_5 - y_2}{y_3 - y_2} P_3 + \frac{y_3 - y_5}{y_3 - y_2} P_2$$

$$P = \frac{x_5 - x_p}{x_5 - x_4} P_4 + \frac{x_p - x_4}{x_5 - x_4} P_5$$

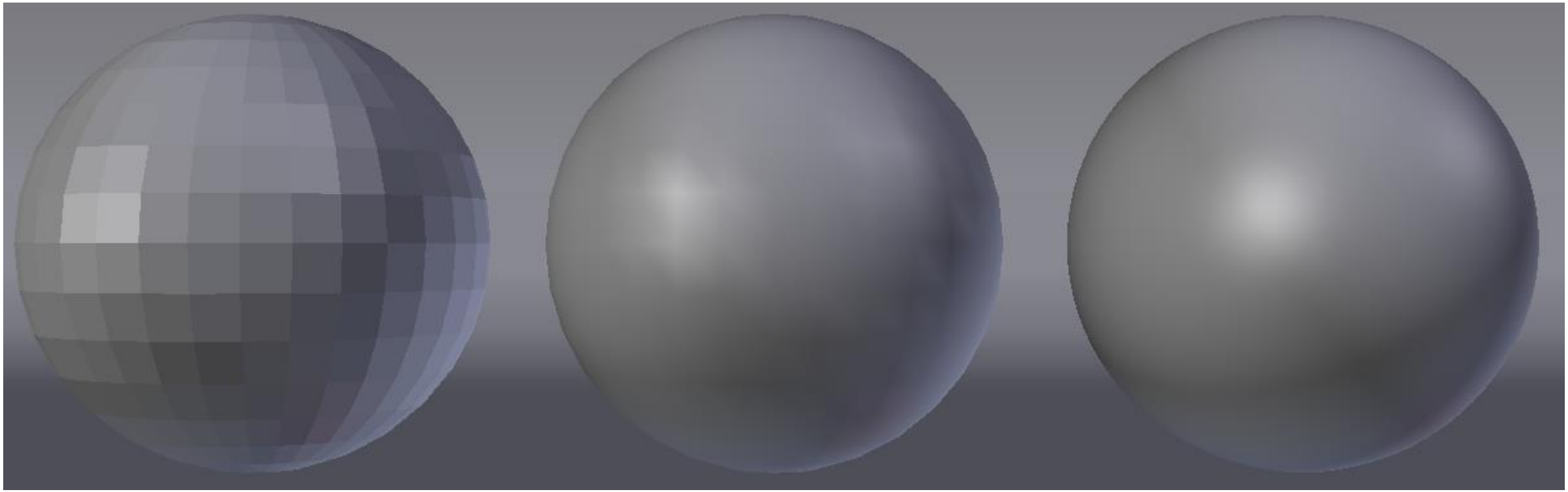
Shading Models

- Phong Shading
  - Similar idea than Gouraud shading, but instead of interpolation of the colour intensities normal vectors are interpolated
- Basic Algorithm
  - Calculate the vertex normal vectors if not already stored or available (this is achieved by averaging the normal vector of all adjacent face normals). Same approach as in Gouraud shading
  - Calculate the normal vectors along the edges based on linear interpolation of the vertex normal and normalise them
  - Calculate the colour intensities along an edge based on a local illumination model
  - Calculate the normal vectors along the scan lines inside a polygon based on linear interpolation based on the edge normal with normalisation



Shading Models

- Comparing Flat, Gouraud and Phong Shading

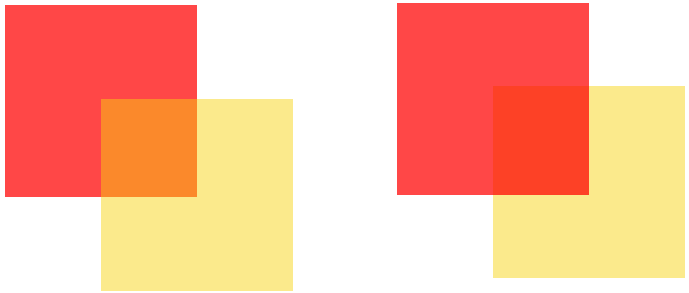


http://www.dare2cre8.com/d2c-004-was-sind-shader-und-materialien/

Transparencies

- Light rays are not only reflected or absorbed they can also be transmitted by some materials - these materials are called translucent
- Simple models ignore refraction of the light

- I_p is the pixel colour we are looking for and it is generated by two overlapping polygons P_1 and P_2
- Variety of mechanisms exist to calculate the pixel colours of translucent overlapping polygons

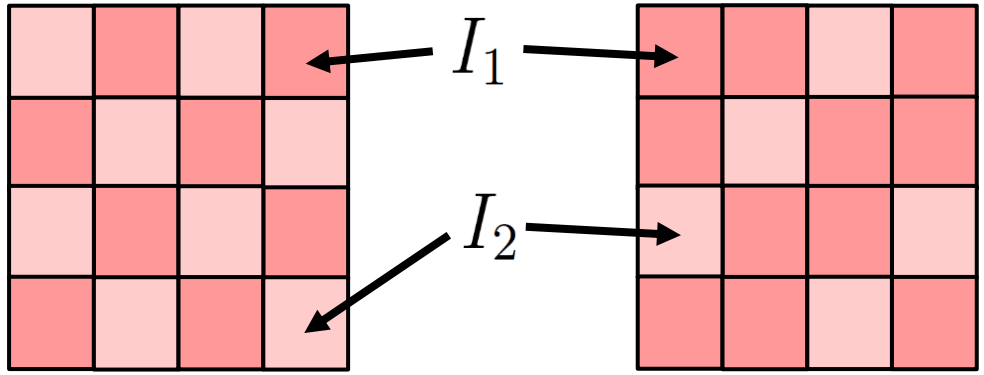


Transparencies

- Interpolated transparency
  - Interpolates colours between two polygons
    $I_p := (1 - k_t) * I_1 + k_t * I_2$
  - I_1 and I_2 are the intensities of colours generated by the local illumination model at the desired pixel of the polygons P_1 and P_2
  - k_t is the transmission coefficient describing how transmissive the material of P_1 is
  - If k_t = 1 then P_1 is fully transmissive and not visible at all (“invisible”)
  - If k_t = 0 then P_1 is not transmissive at all and fully covers P_2 in the area of the overlap
- Filtered transparency
  - Additionally considers the wavelength λ as an additional parameter for the transmission coefficient
    $I_{p\lambda} := (1 - k_t(\lambda)) * I_1 + k_t(\lambda) * I_2$
  - Only light of given wavelengths can be transmitted through the transmissive surfaces

Transparencies

- Screen door transparency
  - Depending on the Amount of transparency different number of pixels are attributed to either I_1 or I_2
  - $I_p \in \{I_1, I_2\}$
  - If high resolution is available its possible to offset the pixels in a given colour
  - Different patterns possible
  - Potential solution in printing if limited amount of colours is available
  - Can lead to problems if multiple polygons are to be overlapping and generate patterns



- More on transparencies in global illumination section

Bibliography

- Kajiya, J. T. The rendering equation ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1986, 20, 143-150
- Johann Heinrich Lambert, Ernst Anding: Lamberts Photometrie: 1. Th. Das directe Licht. - 2. Th. Die Schwächung des Lichts durch durchsichtige Körper, besonders durch Glas. W. Engelmann, 1892
- Phong, B. T. Illumination of Computer-Generated Images, Department of Computer Science, University of Utah, UTEC-CSs-73-129, July 1973.
- Blinn, J. F. Models of light reflection for computer synthesized pictures ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1977, 11, 192-198
- Oren, M. & Nayar, S. K. Generalization of Lambert's reflectance model Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94, ACM Press, 1994
- Torrance, K. E. & Sparrow, E. M. Theory for Off-Specular Reflection From Roughened Surfaces* Journal of the Optical Society of America, The Optical Society, 1967, 57, 1105

Bibliography

- Nicodemus, F. E. Directional Reflectance and Emissivity of an Opaque Surface Applied Optics, The Optical Society, 1965, 4, 767
- Cohen, M. F. Radiosity and Realistic Image Synthesis Morgan Kaufmann, 2016
- Hanrahan, P. & Krueger, W. Reflection from layered surfaces due to subsurface scattering Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘93, ACM Press, 1993
- Debevec, P.; Hawkins, T.; Tchou, C.; Duiker, H.-P.; Sarokin, W. & Sagar, M. Acquiring the reflectance field of a human face Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘00, ACM Press, 2000
- Gouraud, H. Continuous Shading of Curved Surfaces IEEE Transactions on Computers, Institute of Electrical and Electronics Engineers (IEEE), 1971, C-20, 623-629
- Phong, B. T. Illumination for computer generated pictures Communications of the ACM, Association for Computing Machinery (ACM), 1975, 18, 311-317

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Computer Graphics

Texture Mapping and Buffers

FH-Prof. Dr. techn. Dipl.-Inf. (FH) Christoph Anthes, MSc

Professor of Augmented and Virtual Reality

Overview

- Textures
  - Texture coordinates and mapping
- Texture Maps
  - Bump Mapping, Normal Mapping, Displacement Mapping
  - Environment or Reflection Mapping
- Generating Textures
  - Perlin Noise
  - MIP Mapping
  - Texture Packing
- Buffers
  - Z-Buffer or Depth Buffer
  - Shadow Mapping

Textures

- With shading colourful images of geometries can be created but often more detail is desired
- A texture in general is used to add surface details to a 3d model
- That could be hatching, patterns, whole images but also other attributes like wrinkling of the surface, terrain, reflections of the environment, transparencies

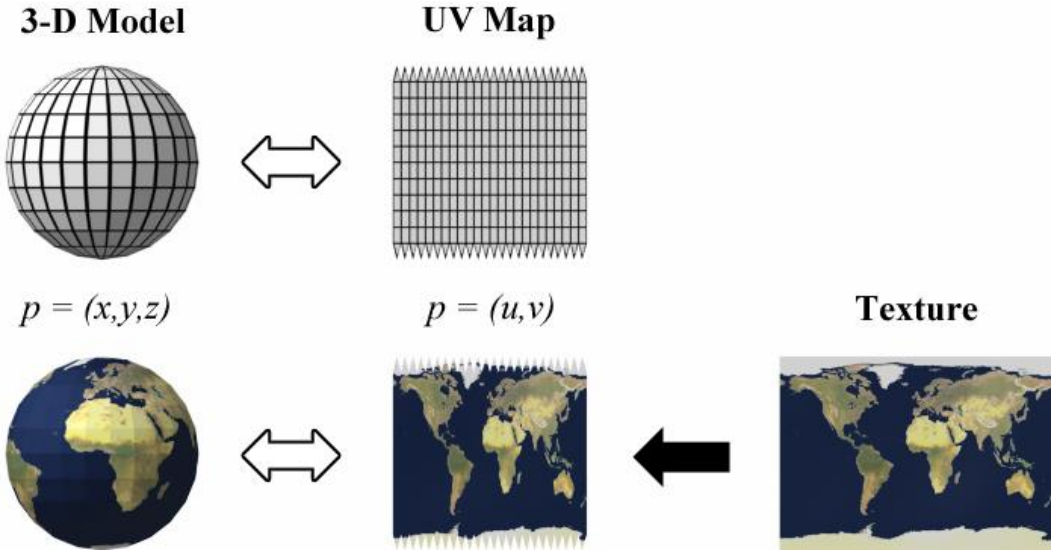
http://media.indiedb.com

- 2d images are used and mapped onto a 3d geometrical object
- Think of it as shrink wrap or gift wrapping paper
- Parts of the texture could be mapped individually only on parts of the object

https://en.wikipedia.org/wiki/Texture_mapping#/media/File:Texture_mapping_demonstration_animation.gif

Texture Mapping

- Texture mapping stores the texture to be applied in a matrix structure (texture map) which is composed by individual texture elements the **texels**
- Each texel can contain for example intensity or colour values
- The 2d coordinate axes of a texture map are typically denoted by u,v or s,t



https://upload.wikimedia.org/wikipedia/commons/0/04/UVMapping.png

- The rectangular texture map is the applied via a projection from a two-dimensional space on a given surface in a three-dimensional space

Texture Mapping

- Mapping from texture space to screen space

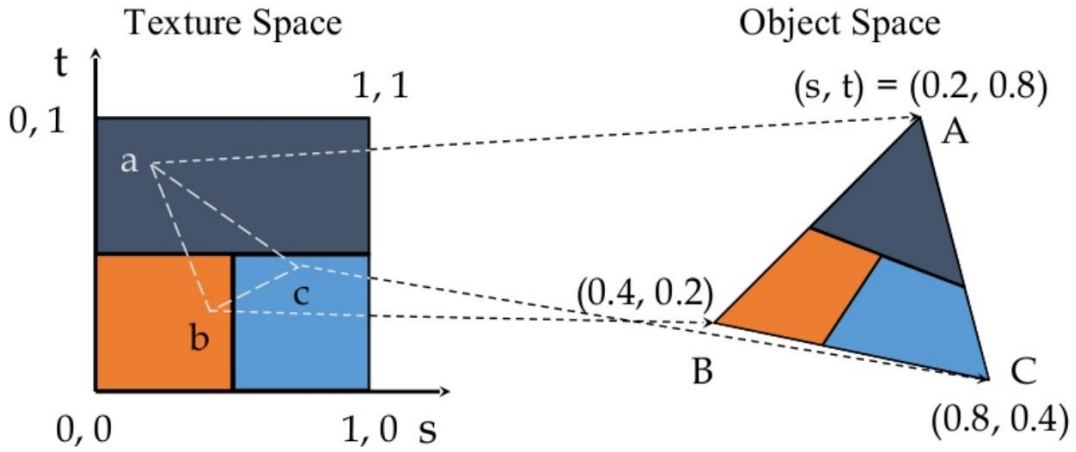
Texture space (s,t)

↓

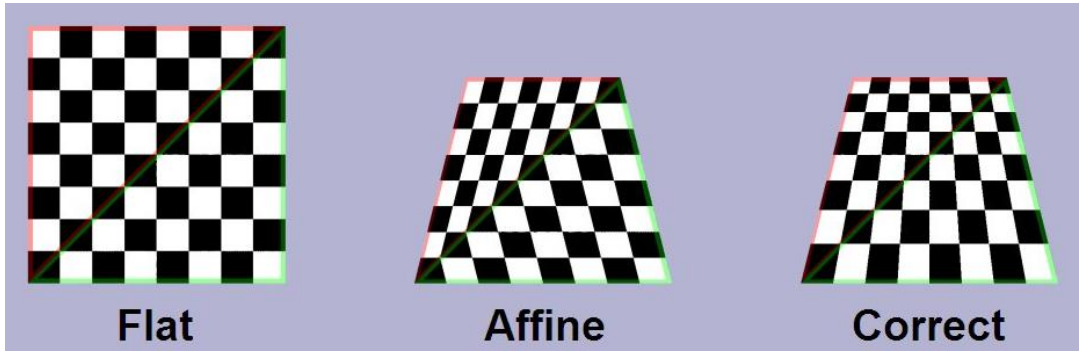
Object space (x_o,y_o,z_o)

↓

Screen space (x,y)



- During rasterisation texture can be considered, which can lead to multiple problems
  - Aliasing artefacts
  - Perspective distortion



https://upload.wikimedia.org/wikipedia/commons/5/57/Perspective_correct_texture_mapping.jpg

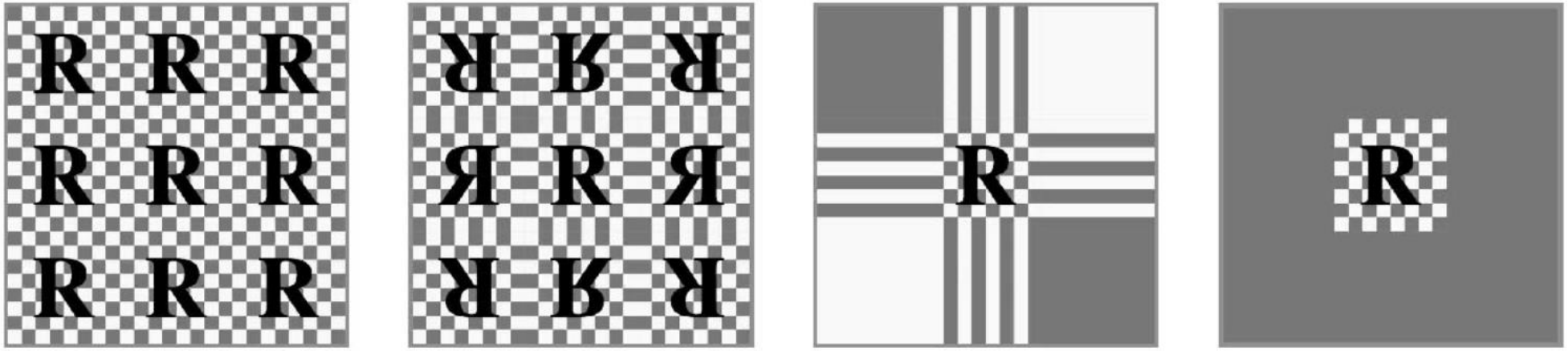
Texture Mapping

- Mapping on a sphere
  - If we assume that the sphere is aligned on the y-axis
  - We can take the d as a unit vector pointing from the point to be textured to the centre of the sphere

$$u = 0.5 + \frac{\arctan 2(d_z, d_x)}{2\pi}$$

$$v = 0.5 - \frac{\arcsin(d_y)}{\pi}$$

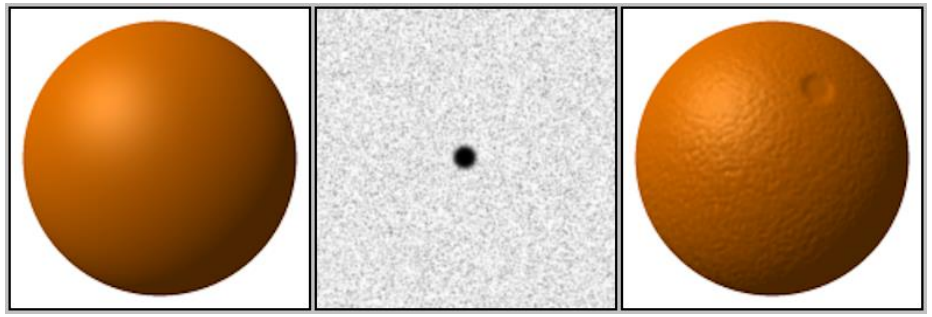
- If the texture coordinates are out of the bounds of 0 and 1 the following operation can be performed
  - Repeating
  - Mirroring
  - Clamping
  - Border



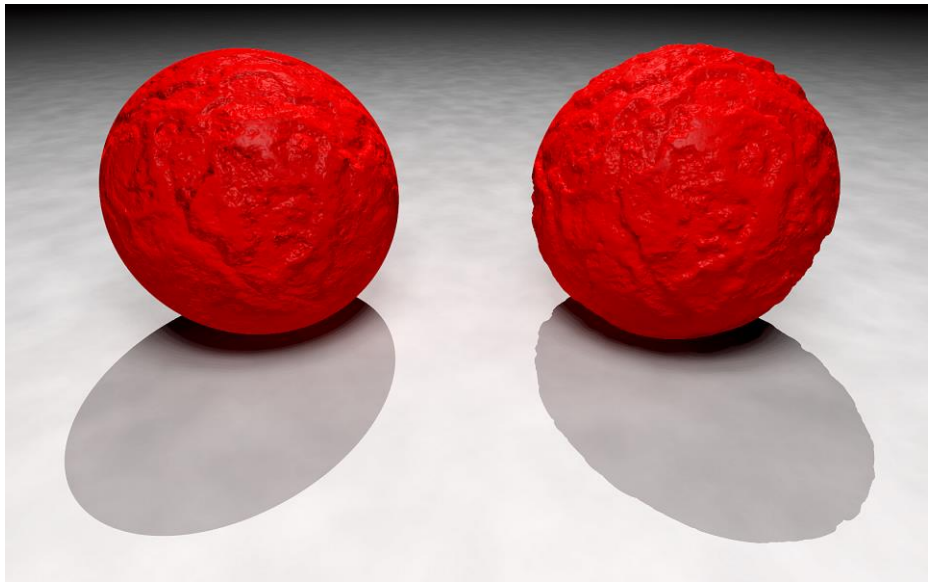
Akenine-Möller, T.; Haines, E. & Hoffman, N. Real-time Rendering Taylor & Francis Ltd., 2008

Bump Mapping

- Idea to represent detailed structure of the surface of an object by providing it as a texture instead of high amount of vertices
- Texture is not used for colour details but for geometrical details
- Texture consists of greyscale values (typically 8-bits e.g. black identifies a dent and white identifies a bulge
- Resulting image provides modification of colour values only, the geometry of the object is not affected
- Silhouette of the object is not affected as well

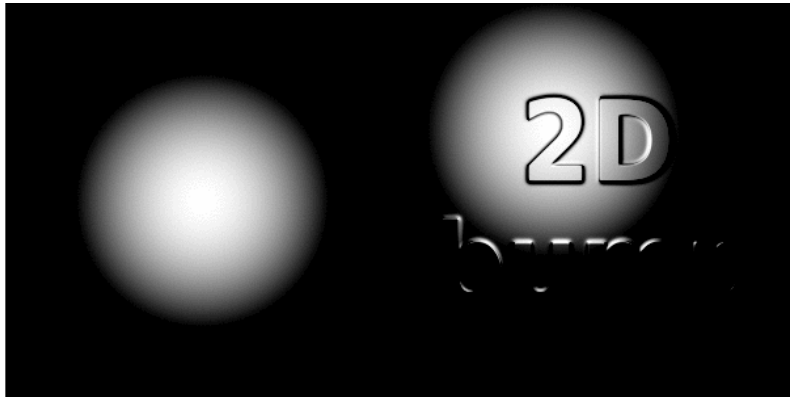


https://upload.wikimedia.org/wikipedia/commons/0/0a/Bump-map-demo-full.png



https://upload.wikimedia.org/wikipedia/commons/4/4e/Bump_map_vs_isosurface2.png

Bump Mapping

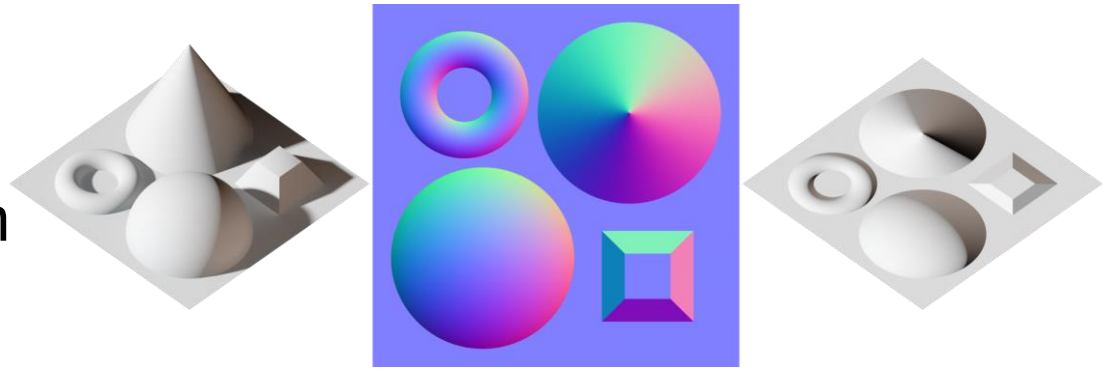


https://en.wikipedia.org/wiki/Bump_mapping#/media/File:FakeBump2D-animation.gif

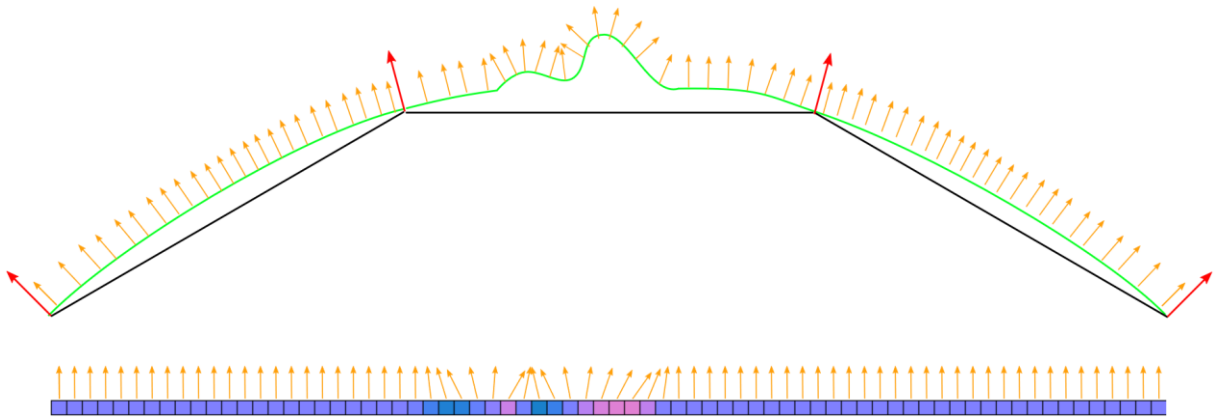
- It can be considered as an additional height map
- In terms of rendering we follow the algorithm
  - Select the appropriate texel intensity value for the given position
  - Calculate the surface normal
  - Combine the calculated surface normal with the actual geometrical surface normal of the object mapped with the bump map
  - Apply your chosen reflection model on the previously combined surface normal
- Improvement is parallax mapping

Normal Mapping

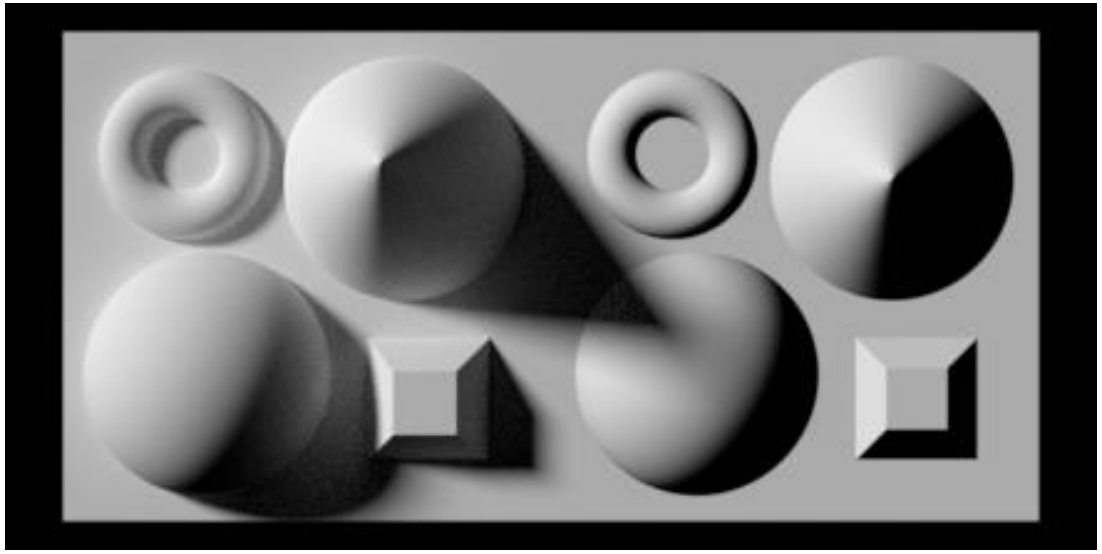
- Similar to bump mapping, can be considered a natural extension
- Normals do not have to be calculated as in bump maps
- They are provided inside the texture
- Each texel provides R,G,B information which is mapped on the x,y,z coordinates



https://en.wikipedia.org/wiki/Normal_mapping#/media/File:Normal_map_example_with_scene_and_result.png



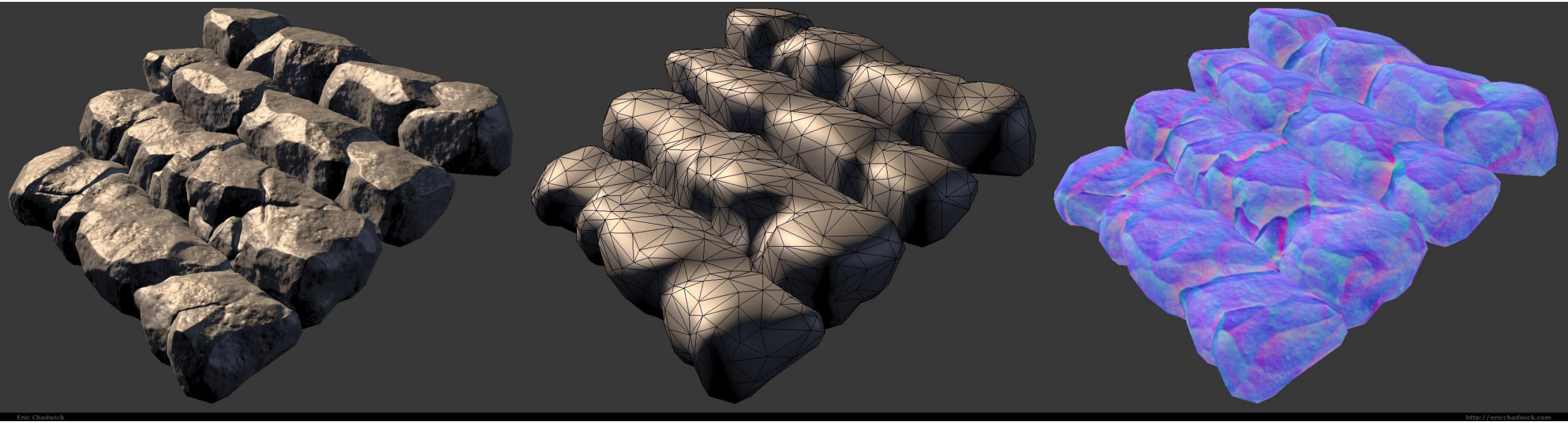
https://docs.unity3d.com/Manual/StandardShaderMaterialParameterNormalMap.html



https://upload.wikimedia.org/wikipedia/commons/e/e4/Rendering_with_normal_mapping.gif

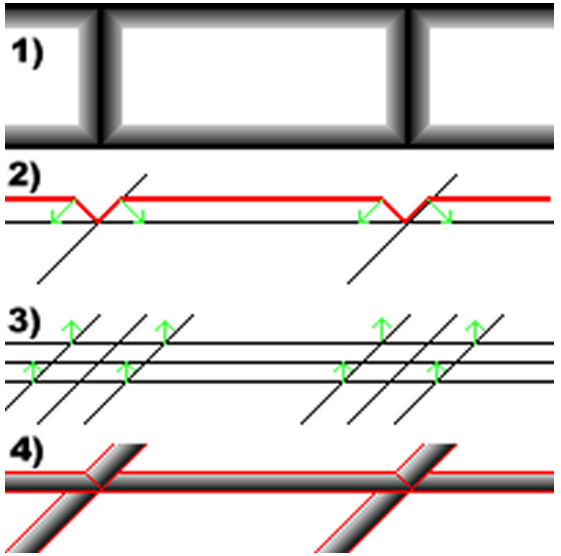
Normal Mapping

- Example of a high quality normal map allied on a low-poly structure



http://wiki.polycount.com/w/images/c/c9/Normalmap_stairs.jpg

Displacement Mapping

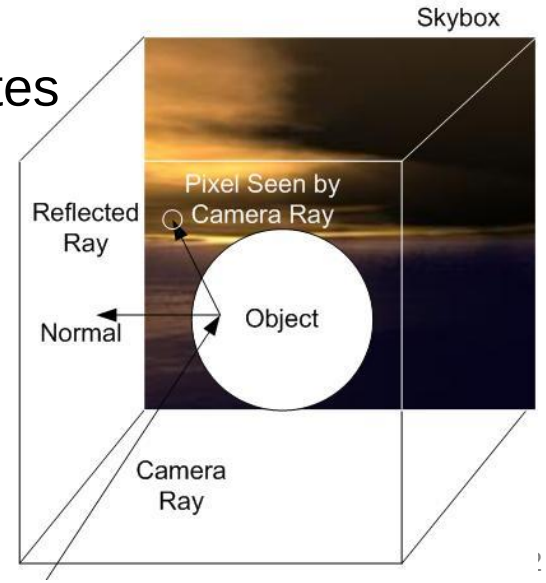


- While normal and bump mapping do not affect the geometry of the object, displacement mapping actually does modify the geometry
- Thus it affects render complexity significantly also silhouettes are affected
- Steps of displacement mapping
  1. Height map contains greyscale values (similar to bump map)
  2. Geometry is subdivided based on the height map which is applied with regular texture mapping
  3. Transformation of the newly subdivided grid is performed based on the normal vectors at the position of subdivision edges with the depth of the height map
  4. Newly generated structure resembles the displacement on original geometry
- Can be efficiently implemented in a depth buffer

Environment Mapping or Reflection Mapping

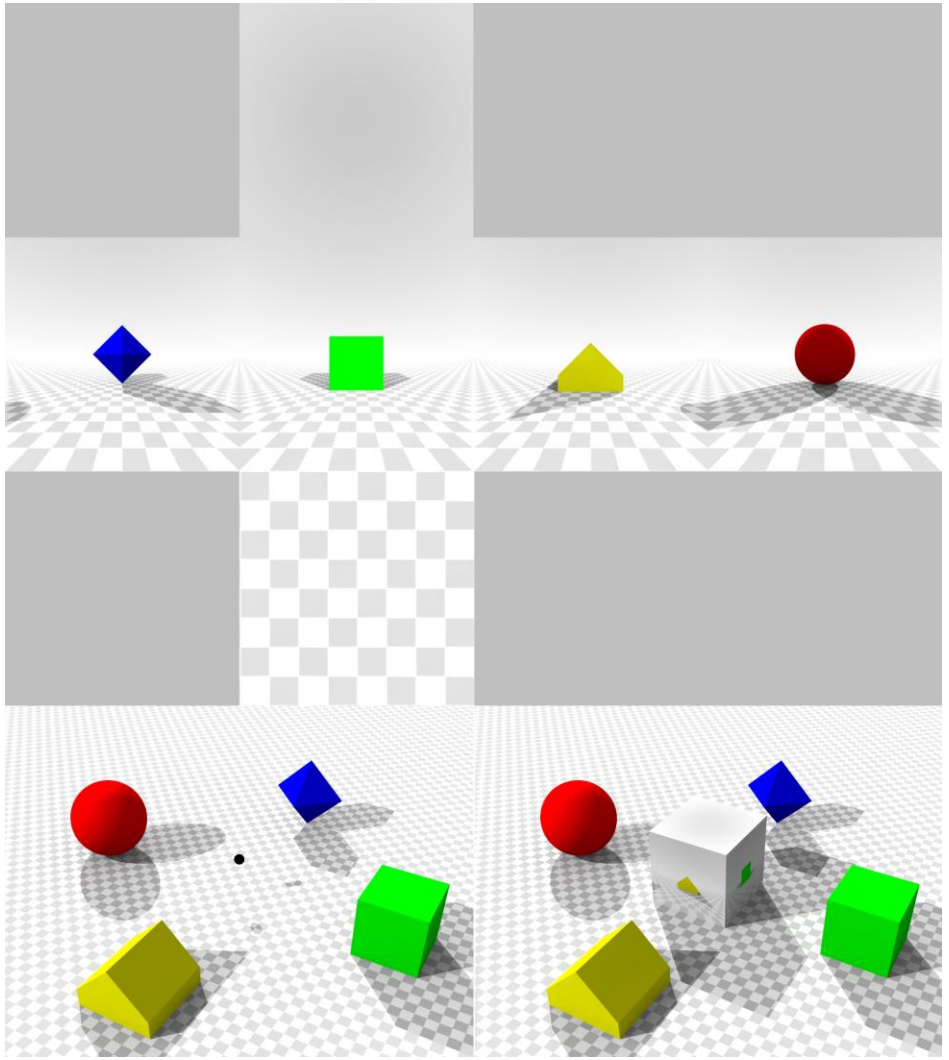
- Used to display reflections on objects in the scene
- Generating a reflection map
  - The environment from the position of the object where the mapping is to be applied is rendered or recorded
  - An inside-out view is given on the map
  - It then is mapped on a sphere or a cube
- Accessing the corresponding map value
  - At point of interest the reflection map is indexed by the polar coordinates of the reflected view vector around the surface normal at the point
  - Same vector R from Phong shading
- Problem
  - Reflection map is considered an infinitely large sphere since only direction is taken into account not position of object inside sphere

http://tfc.duke.free.fr/screens/envmap.png



Environment Mapping or Reflection Mapping

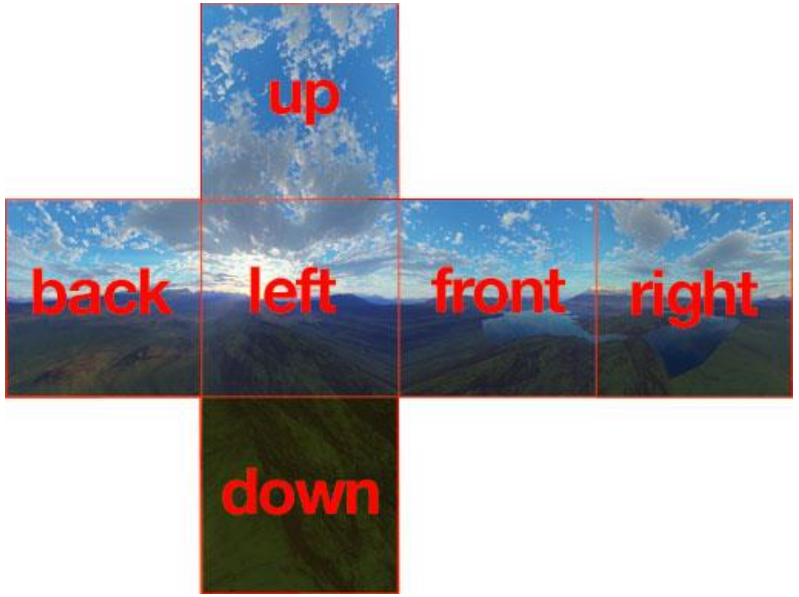
- Sphere Mapping
  - Texture depicting what a mirrored sphere would look like if it were placed into the environment
  - Orthographic projection is used
  - The coordinate range from 0° to -360° longitude and -90° to 90° latitude
- Cube Mapping
  - Instead of spherical mapping the reflection map is generated by 6 images represented in a cubic shape
- In case only one surface should be reflecting scene could be rendered from the viewpoint of the surface



https://upload.wikimedia.org/wikipedia/commons/e/e2/Panorama_cube_map.png

Environment Mapping or Reflection Mapping

- Sky boxes and sky domes
  - Used for rendering surroundings of a scene to avoid modelling it and thus reducing the amount of required polygons
  - Typically rendered into the background using the frame buffer
  - Can also be mapped on a box or a dome moving along with the camera while the camera is placed in the centre of the skybox
  - Some applications move the skybox at a lower speed than the camera allowing the camera to get closer to the skybox
  - This counteracts the impression that the background is infinitely far away

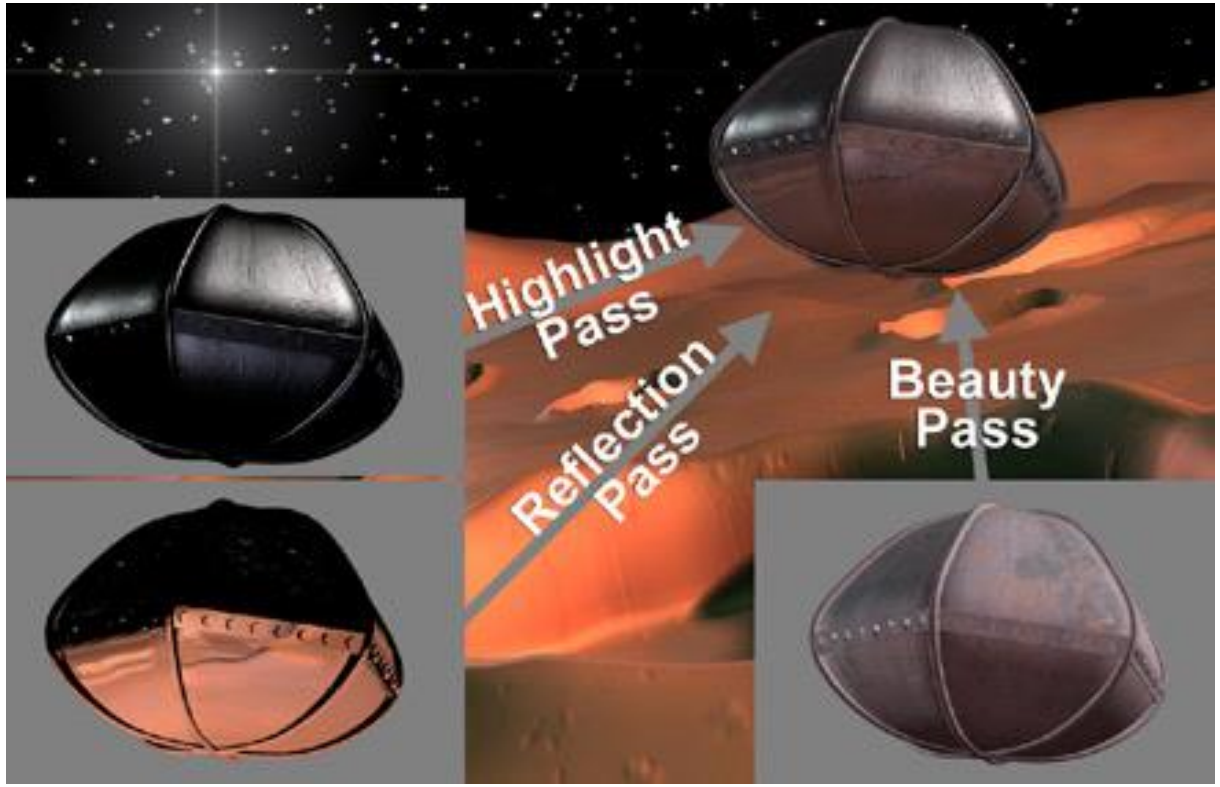


https://en.wikipedia.org/wiki/Skybox_(video_games)#/media/File:Skybox_example.png

https://upload.wikimedia.org/wikipedia/commons/0/06/Skybox_sky.png

Multi-Pass Rendering

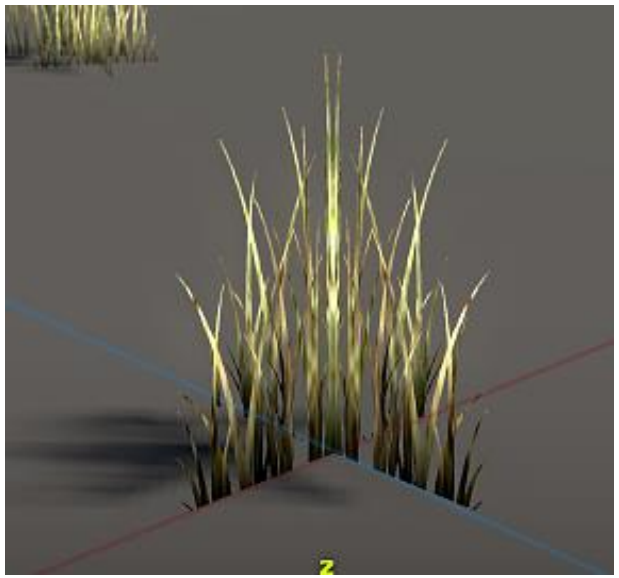
- Combination of multiple textures is common to safe texture memory and rendering performance
- Camera settings are kept and scene is rendered multiple times in different back buffers
- The results of the rendering passes can be combined with alpha blending for example
- When output from one render pass is taken as input into a second render pass we speak of multi-pass rendering



http://www.3drender.com/light/compositing/index.html

Billboards

- Simple planar representations of often complex geometrical objects
- The orientation of a billboard is always perpendicular to the view vector of the camera
- Are often used with particle systems (more in animation part of the lecture)
- Billboard clouds are often used to generate plants like grass and trees in computer games, in this case the orientation depends of the orientation of the object
- They could also be used in a crossed way to generate a more realistic look when changing the perspective

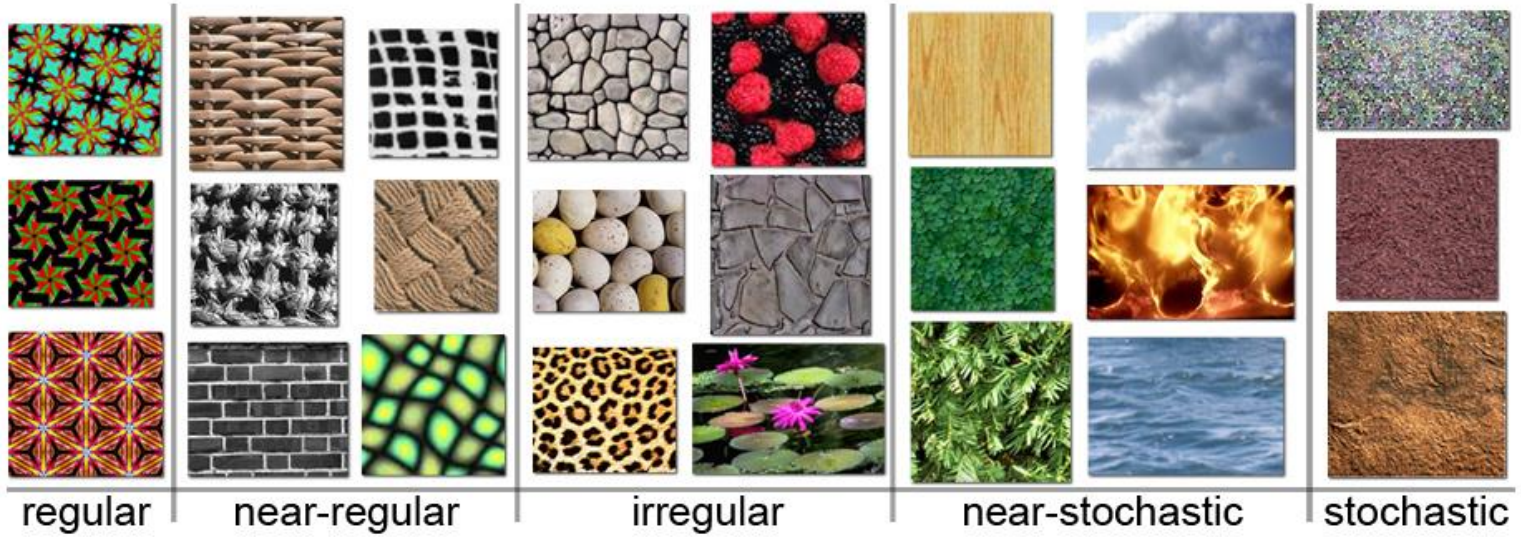


https://forum.unity.com/attachments/upload_2017-4-26_12-32-18-png.228601/

Generating Textures

- Can be scanned, painted, generated procedurally or rendered in an earlier render pass
- Example Quilting
  - Generating textures based on similarity in a larger source area

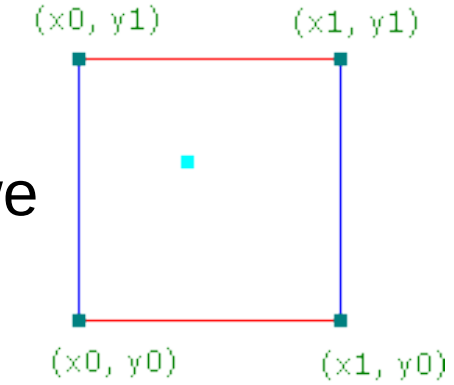
https://de.wikipedia.org/wiki/Textursynthese#/media/File:Imagequilting.gif



https://de.wikipedia.org/wiki/Textursynthese#/media/File:Texture_spectrum.jpg

Generating Textures - Perlin Noise

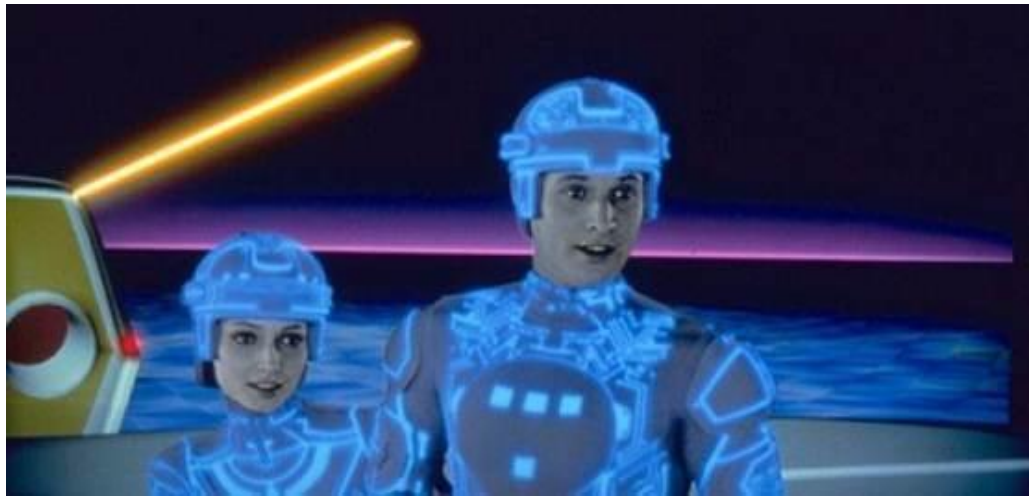
- Is a stochastic Noise function
- Use for procedural texturing, including clouds, waves, tornadoes, rocket trails, heat ripples, incidental motion of animated character
- We have a regular grid where an element ranges from (x_0,y_0) to (x_1,y_1)
- In-between we have a point (x,y) based on which we want to calculate the noise value



https://mzucker.github.io/html/perlin-noise-math-faq.html

https://en.wikipedia.org/wiki/Perlin_noise#/media/File:Perlin_noise.jpg

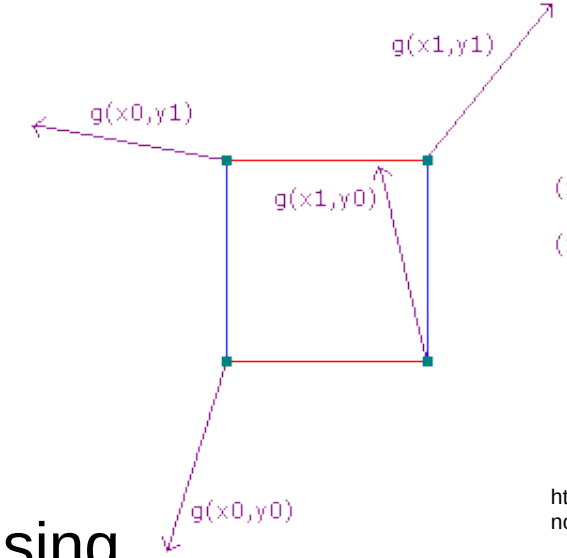
https://en.wikipedia.org/wiki/Perlin_noise#/media/File:Fractal_terrain_texture.jpg

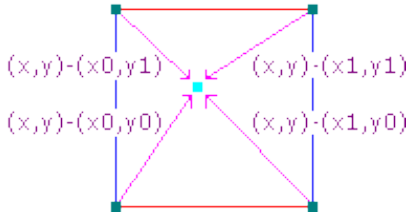


https://thebookofshaders.com/11/

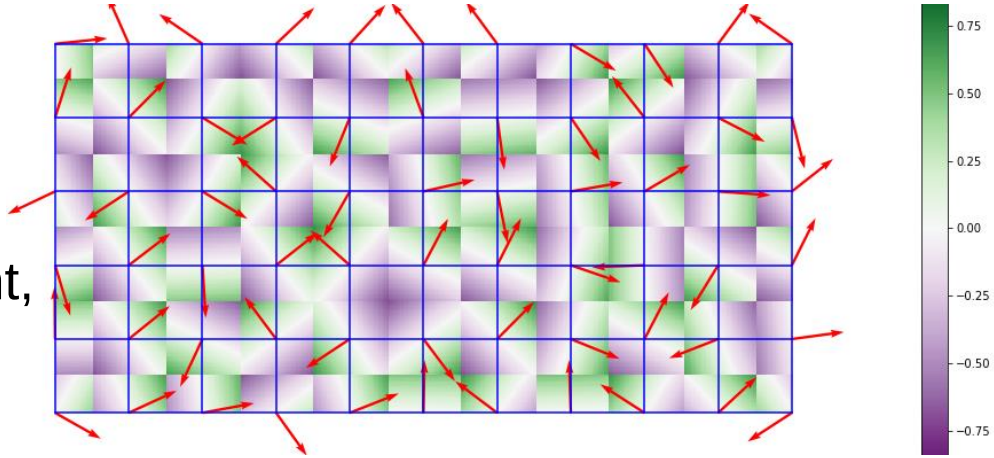
Generating Textures - Perlin Noise

- We additionally calculate for each grid point a pseudo-random gradient g (the gradient is always the same for the same grid point) with the length of one
- Additionally we calculate a vector from each grid point to our desired position (x,y)
- We calculate the influence of each gradient by using the scalar product between each of the vectors pointing to (x,y) and the associated gradient
- Graphically this means
  - if vectors are perpendicular no influence from gradient,
  - positive when in direction of the gradient
  - and negative when opposite direction





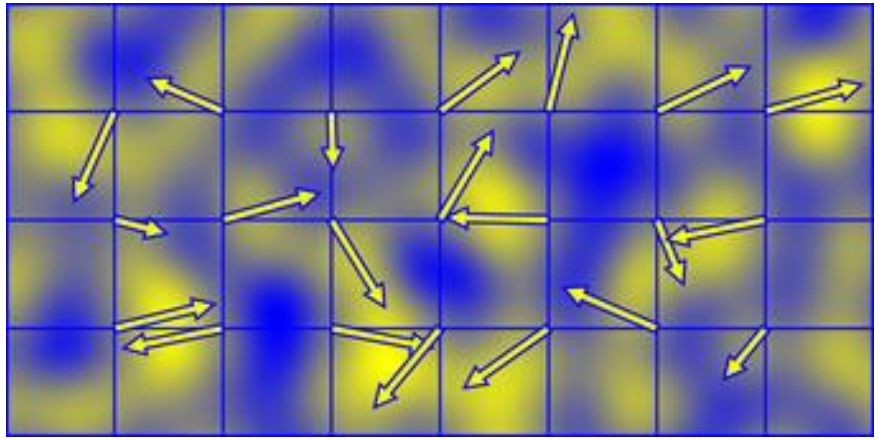
https://mzucker.github.io/html/perlin-noise-math-faq.html



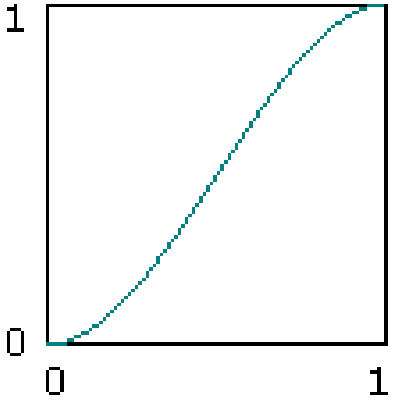
https://en.wikipedia.org/wiki/Perlin_noise#/media/File:PerlinNoiseDotProducts.png

Generating Textures - Perlin Noise

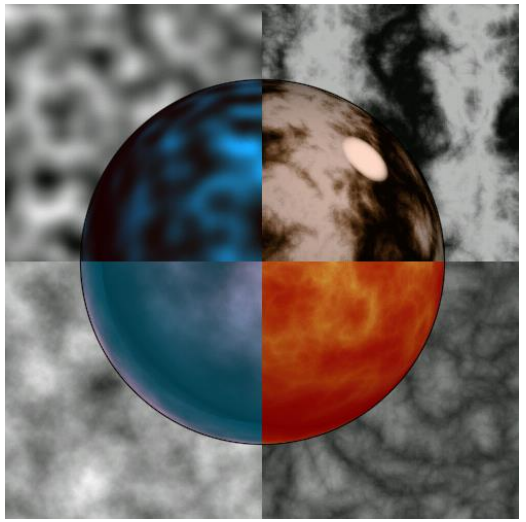
- Form this we retrieve influence values for each corner
- In the next step we calculate averages between two influence values on the x-axis
- Then we use the two generated values to average between them on the y-axis
- To avoid getting a grid pattern look, an ease or fade interpolation function is used



https://flafla2.github.io/2014/08/09/perlinnoise.html



https://mzucker.github.io/html/perlin-noise-math-faq.html



https://web.archive.org/web/20071008170055/http://www.noisemachine.com/talk1/19.html

Generating Textures - Texture Baking

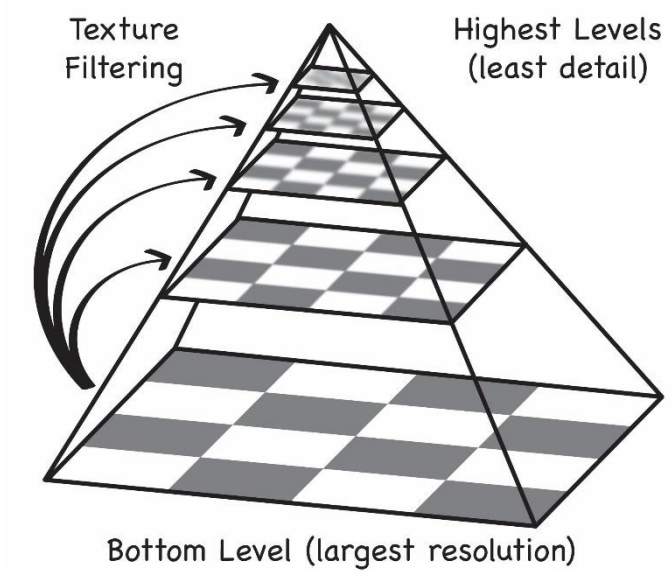
- Textures can also be generated based upon the scene
- Texture baking is the process of pre-computing textures for the scene
- If lighting is static parts of it can be precomputed
  - As input for the backing process the scene is textured with static materials etc.
  - Additional high quality lighting (global illumination like raytracing) inside the textured scene is calculated offline
  - The results are stored inside textures that contain the lighting information
  - These textures can be used in combination with the material textures (multi texturing) to generate a realistic look of the scene
- In case the scene changes or the light source attributes change texture baking can’t be used
- Also normal maps, shadow maps etc. can be generated

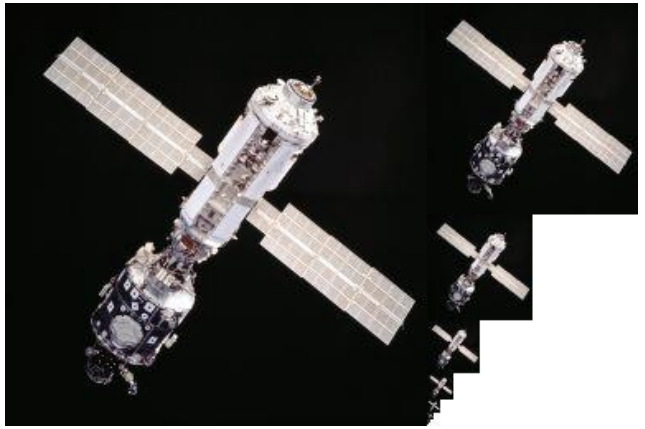
MIP Mapping

- MIP (Multum In Parvo) Mapping
  - Idea of level-of-detail applied on textures instead of geometry
  - Not used for render performance rather used to reduce aliasing artefacts
  - The same texture is reproduced multiple times at half the resolution of the previous texture (e.g. orig. 256x256 MIP map 128x128, 64x64, 32x32)
  - Increased storage space for MIP maps is at most 1/3 of the original space (geometric series)

$$\sum_{i=0}^{\infty} \left(\frac{1}{4}\right)^i = \frac{1}{1-\frac{1}{4}} = \frac{4}{3} = 1 + \frac{1}{3}$$

  - Depending on the distance lower or higher resolution MIP maps are displayed reducing aliasing problems in rasterisation
  - Variety of filtering approaches exist for downsampling a MIP map





https://upload.wikimedia.org/wikipedia/commons/5/5c/MipMap_Example_STS101.jpg

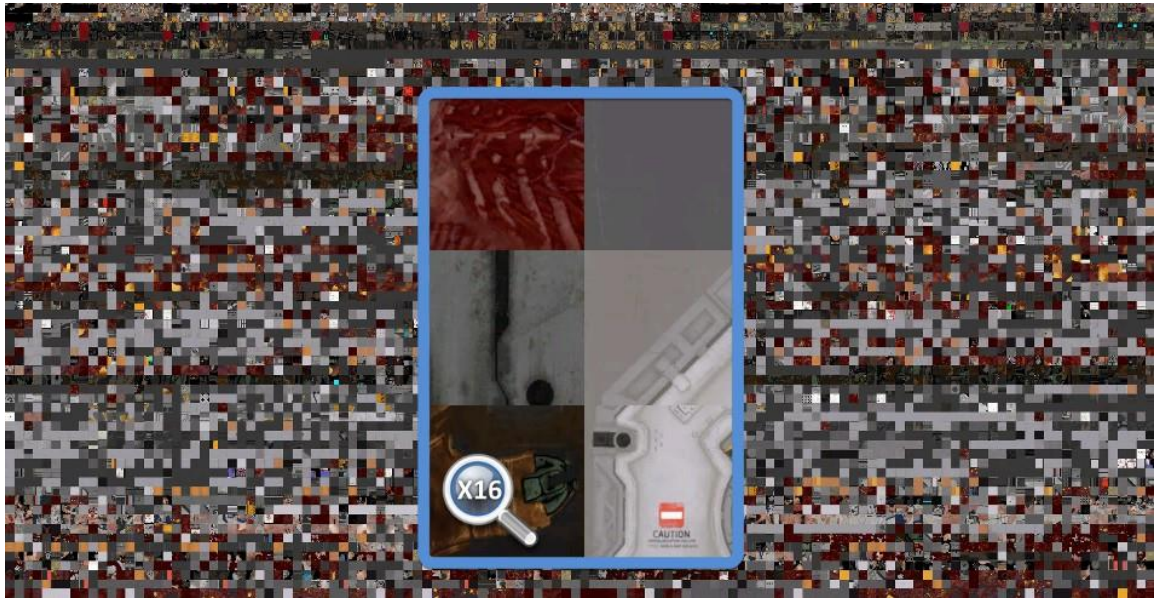
Texture Packing

- In general textures are either dynamically generated or mapped with the help of a 3d modelling tool like 3DS Max or Cinema 4d
- Typically the textures are in sizes of powers 2 for better memory storage although they can be compressed on disc they are uncompressed in memory
- Textures can be sorted into individual files but typically textures are stored inside atlases, a large texture consisting of many individual textures or parts of textures
- Bin packing algorithms can be used to optimise these texture atlases

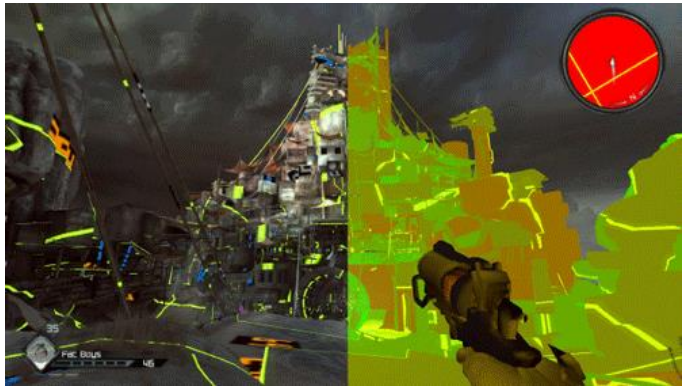
Odaker, T.; Wiedemann, M.; Anthes, C. & Kranzlmüller, D. Texture analysis and repacking for improved storage efficiency Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology - VRST '16, ACM Press, 2016

Texture Packing

- Mega textures
  - Idea to map every detail in an environment and to avoid tiles allowing artists more freedom
  - Introduced in Rage (id Software)
  - Based on clip texturing
  - Texture segments are streamed dynamically from disc or memory on the graphics board
  - Texture sizes of up to 128000×128000 pixels can be used



http://www.adriancourreges.com/img/blog/2016/doom2016/shot/05_megatext_illus.jpg



http://renderingpipeline.com/wp-content/uploads/2012/03/virtualtextureloading.gif

Rendering Textures

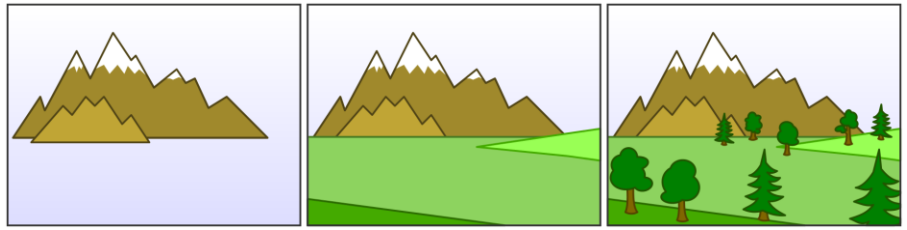
- During rasterisation, for each rendered pixel of the textured object we need to look up a color value from the texture
  - Will almost always fall between texture pixels (texels)
  - Texture may have too much resolution: sampling or integration
  - Texture may have too little resolution: interpolation
- Naïve approach: pick the nearest neighbor pixel
  - Leads to blocky textures
- Better approach: bilinear filtering
  - Pick the 4 neighboring pixels and linearly interpolate
- Trilinear filtering: find the 2 best levels of the MIP map and interpolate within and between them

Buffers on Graphics Boards

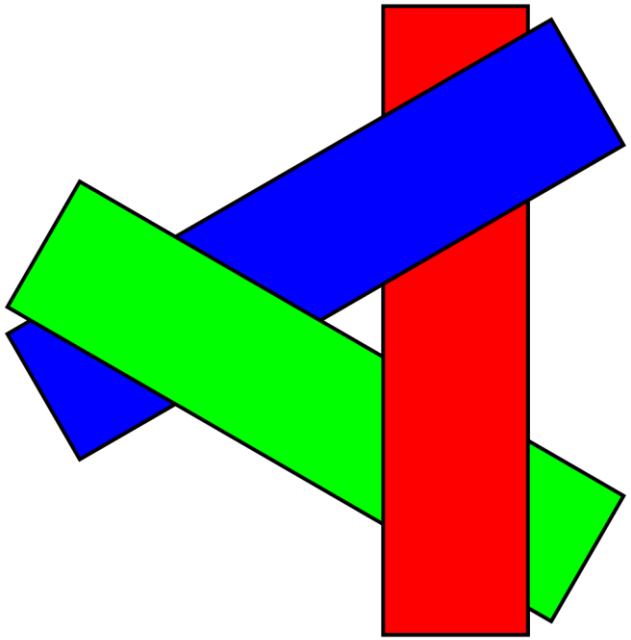
- Buffers are not specific to a graphics board although many of them are commonly implemented in graphics APIs
- The basically reserve an area of memory of a defined precision (bits) and resolution (x,y)
- We have heard of so far
  - Frame buffer separated in
    - Front buffer (used for rendering)
    - Back buffer (used for double-buffering) implemented with page flipping
  - Depth buffer (not in detail yet, more on the next slide)
- Other common buffers
  - Stencil buffer
  - Accumulation buffer (used for blur effects for example)

Dealing with Occlusion

- Depth sort
  - Sort polygons according to their distance to the near clipping plane for drawing
- Painter's algorithm
  - Basic idea to draw elements which are in the background first
  - Follow with the next layer and overdraw existing polygons if necessary
  - Front polygons will stay visible
- Problems
  - Self occlusion (not with primitives)
  - Circular occlusion
  - Intersections



https://de.wikipedia.org/wiki/Maleralgorithmus#/media/File:Painter%27s_algorithm.svg



https://de.wikipedia.org/wiki/Maleralgorithmus#/media/File:Painters_problem.svg

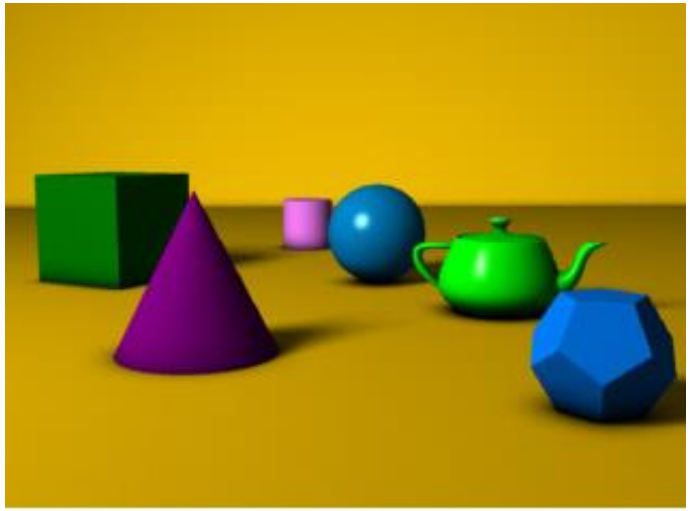
Z-Buffer or Depth Buffer

- Problem with occlusion often not solvable on an object or polygon level
- Approach to compute the depth of an object on a fragment or pixel level
- Additional to a pixel in the regular frame buffer a pixel in the depth buffer or z-buffer is stored as well
- The depth buffer is initialised with the values from the far clipping plane since further values will be ignored

- Extremely cheap to compute, can be used for a variety of tricks
- The z-buffer is typically built into graphics hardware

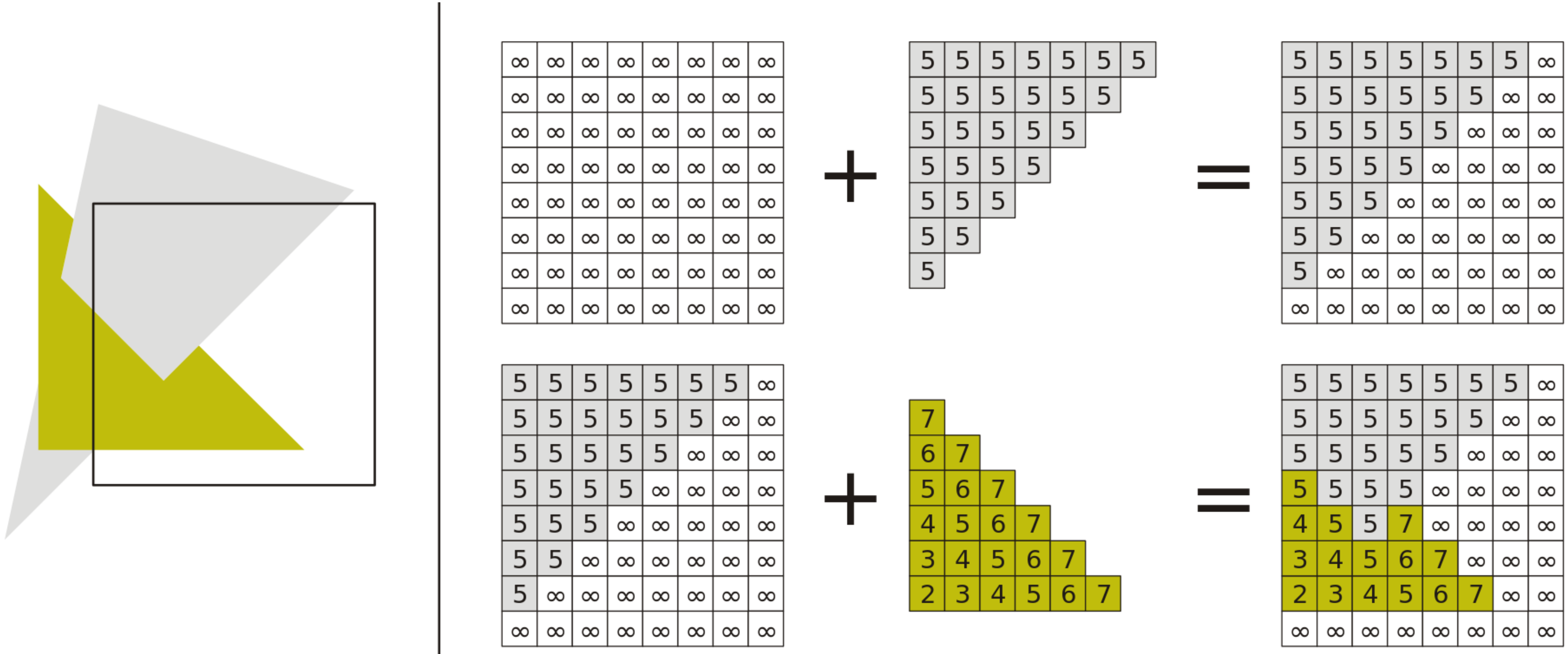
Z-Buffer or Depth Buffer

- Filling the z-buffer
- Algorithm
  - **For each polygon in the Frustum**
    - Determine the pixels to be drawn
    - **For each pixel to be drawn**
      - Compute the depth value z (distance to the camera) at the x,y position
      - If z(x,y) < current value (x,y) then current value (x,y) := z(x,y)
  - **Render image**



https://upload.wikimedia.org/wikipedia/commons/8/8f/Z-buffer_no_text.jpg

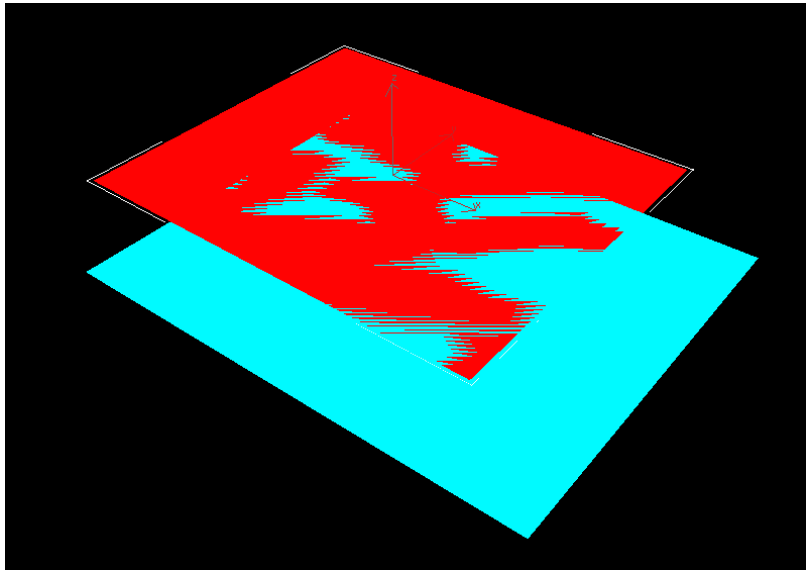
Z-Buffer or Depth Buffer



https://de.wikipedia.org/wiki/Z-Buffer#/media/File:Z-buffer.svg

Z-Buffer or Depth Buffer

- The resolution of the z-buffer is limited (e.g. 8 bit)
  - This can lead to problems with large models
  - If distance between z-buffer units is to large pixels of polygons might be sorted wrong
  - This can lead to flickering (so called z-buffer fighting) or stripped objects
- The z-buffer is not linear - the resolution is higher closer to the camera/near clipping plane and lower to the far clipping plane
  - Typically logarithmic
  - Z-buffer fighting and stripping might appear when objects move further away from the camera
  - The closer the clipping planes are the better the resolution in the frustum, the further they are away from each other the more likely artifacts in terms stripping and z-buffer-fighting might appear



https://en.wikipedia.org/wiki/Z-fighting#/media/File:Z-fighting.png

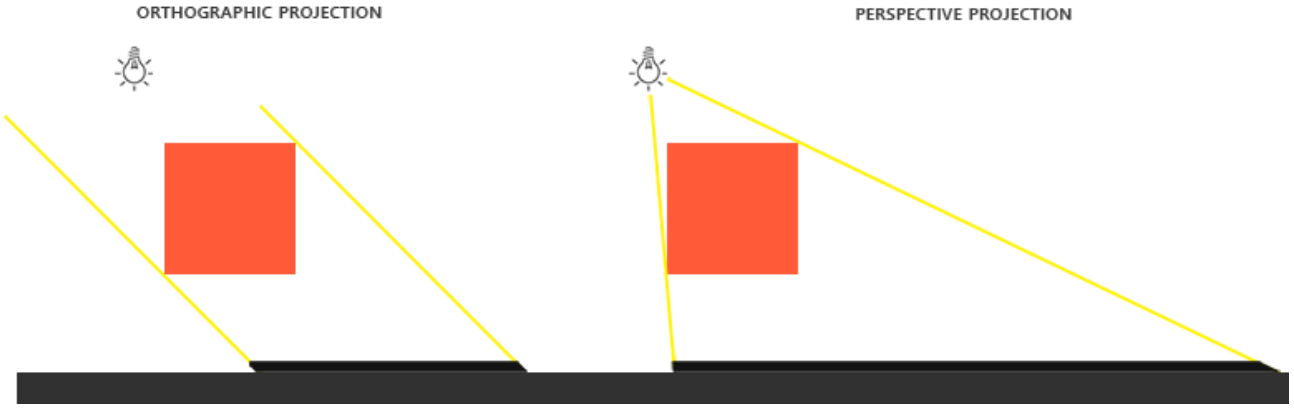
Z-Buffer or Depth Buffer

- Artifacts caused through z-buffer are deterministic
- The same camera position with the same scene leads to the same artifacts
- Avoid issues by placing clipping planes wisely depending on your current scene
- Never place two coplanar polygons at the same location is space (happens quite often, when you work with industry data)
- Try to have a minimum distance between coplanar polygons

- Initialisation of the z-buffer
  - Besides initialising with the depth of the far clipping plane parts of it can be set to near values and then processed, this generates holes in the rendered polygons
  - A second set of polygons can then be used again after re-initialisation to fill the holes
  - These polygons are the visible through the holes

Shadow Mapping

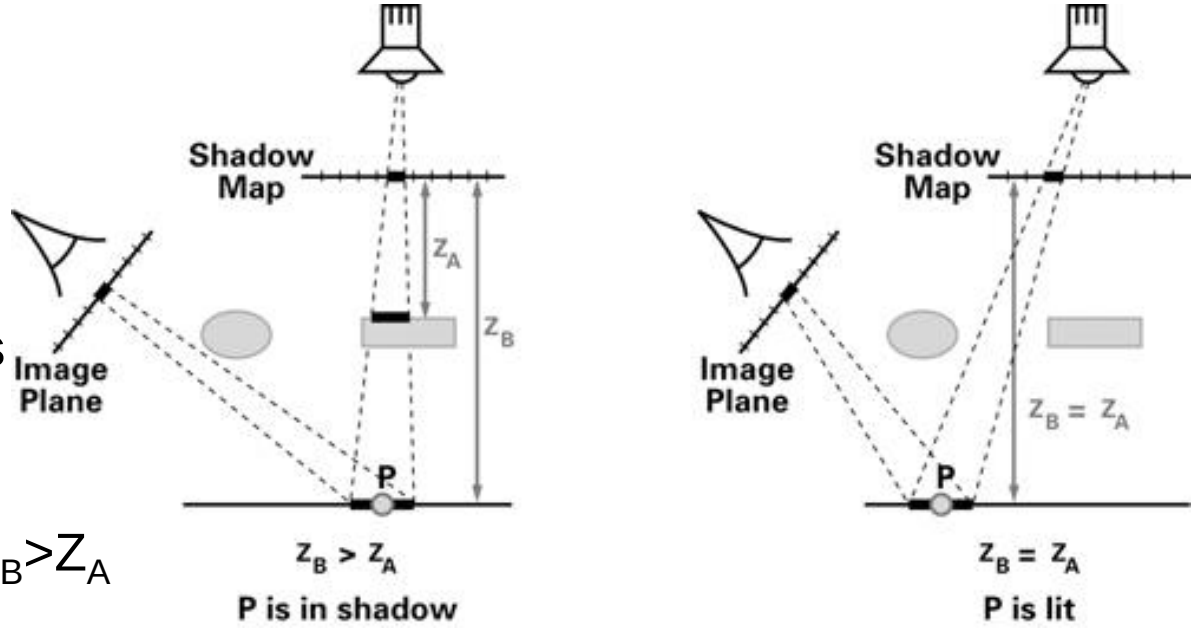
- With local illumination models shadows were not considered so far
- To generate a shadow in the scene shadows are separately calculated and applied
  - From the position of each light source, record a depth buffer (for point or spot lights, perspective projection should be used, for directional light parallel projection should be used)
  - Often only back faces are used since this is sufficient for generating a shadow map
  - In this step all colour or texture calculation is omitted to increase performance
  - If scene or lights change, shadow map must be recalculated
  - For each pixel in buffer, we know how far from the light it is (depth buffer) this is stored inside a texture



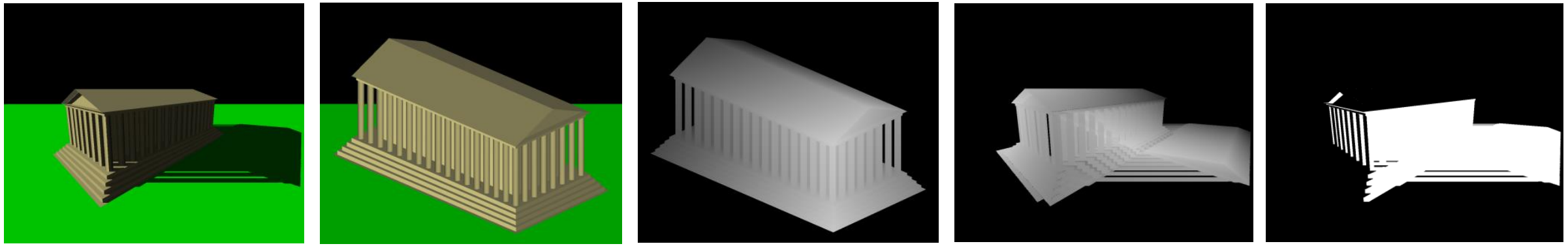
https://learnopengl.com/img/advanced-lighting/shadow_mapping_projection.png

Shadow Mapping

- Rendering a shadow map
  - For each rendered pixel in the camera image, check distance of its surface point to the light source
    - If distance in shadow buffer (Z_A) equals distance to the light source (Z_B): $Z_B=Z_A$ → in this light
    - If distance in shadow buffer is closer: $Z_B>Z_A$ → in the shadow of this light



http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter09.html



https://en.wikipedia.org/wiki/Shadow_mapping

Bibliography

- Akenine-Möller, T.; Haines, E. & Hoffman, N. Real-time Rendering Taylor & Francis Ltd., 2008
- Blinn, J. F. Simulation of wrinkled surfaces ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1978, 12, 286-292
- Kaneko, T.; Takahei, T.; Inami, M.; Kawakami, N.; Yanagida, Y.; Maeda, T. & Tachi, S. Detailed Shape Representation with Parallax Mapping ICAT 2001, 2001
- Cook, R. L. Shade trees Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘84, ACM Press, 1984
- Blinn, J. F. & Newell, M. E. Texture and reflection in computer generated images Communications of the ACM, Association for Computing Machinery (ACM), 1976, 19, 542-547
- Cohen, J.; Olano, M. & Manocha, D. Appearance-preserving simplification Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘98, ACM Press, 1998
- Perlin, K. An image synthesizer Proceedings of the 12th annual conference on Computer graphics and interactive techniques - SIGGRAPH '85, ACM Press, 1985

Bibliography

- Lagae, A.; Lefebvre, S.; Cook, R.; DeRose, T.; Drettakis, G.; Ebert, D.; Lewis, J.; Perlin, K. & Zwicker, M. A Survey of Procedural Noise Functions Computer Graphics Forum, Wiley, 2010, 29, 2579-2600
- Odaker, T.; Wiedemann, M.; Anthes, C. & Kranzlmüller, D. Texture analysis and repacking for improved storage efficiency Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology - VRST '16, ACM Press, 2016
- Williams, L. Casting curved shadows on curved surfaces Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH '78, ACM Press, 1978
- Edwin Catmull: A Subdivision Algorithm for Computer Display of Curved Surfaces. Dissertation, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City 1974

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Computer Graphics

Rasterisation and Global Illumination

FH-Prof. Dr. techn. Dipl.-Inf. (FH) Christoph Anthes, MSc

Professor of Augmented and Virtual Reality

Overview

- Rasterisation
  - Naïve Line Drawing
  - Bresenham
- Anti-Aliasing
  - Gupta-Sproull
  - Wu
- Global Illumination
  - Raytracing
  - Radiosity
  - Volume Rendering

Rasterisation

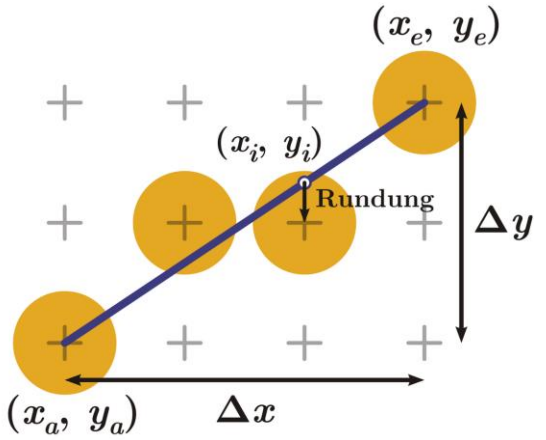
- In the rasterisation or scan conversion step the projected 3D model of the scene resulting in 2D coordinates is rasterised and a 2D image is generated
- The colours from reflection and shading model are evaluated
- Textures are applied
- Fragments in front based on the z-Buffer are to be drawn

- Variety of algorithms exist for optimising rasterisation approach
  e.g. Bresenham

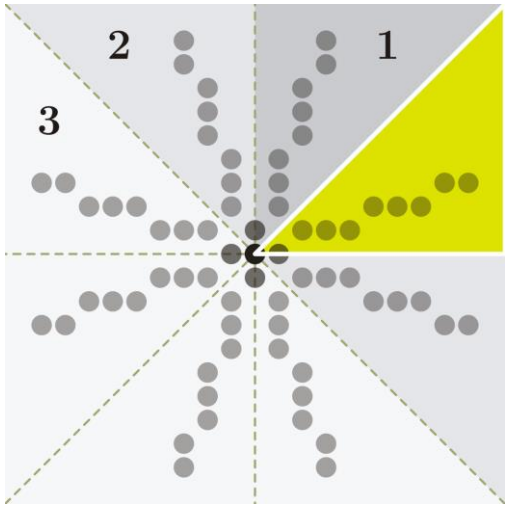
Rasterisation

- Drawing a Line: Naïve Approach
  - Line from (x_a,y_a) to (x_e, y_e),
  - Set $d_x := x_e - x_a$, $d_y := y_e - y_a$, $m := d_y/d_x$
- For x from 0 to d_x do:
  - setpixel ($x_a + x$, $y_a + m * x$)
- Computationally unnecessary expensive - in each step:
  - 1 float multiplication
  - 1 round to integer

- To work with other areas you'll have to slightly adapt
  - Assume -1 < m < 1, otherwise exchange x and y
  - Assume $x_e > x_a$, otherwise switch endpoints



https://upload.wikimedia.org/wikipedia/commons/2/24/Line_scan-conversion_rounding-de.svg



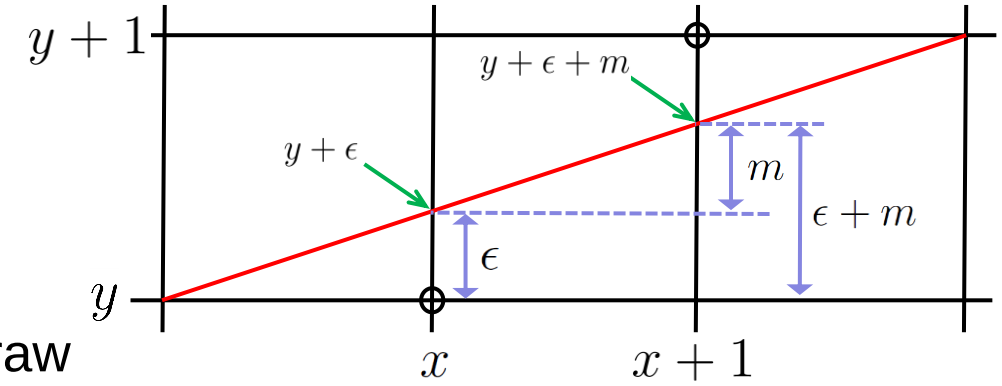
https://upload.wikimedia.org/wikipedia/commons/f/f7/Line_drawing_symmetry.svg

Rasterisation

- Bresenham‘s Algorithm
  - Idea to work in incremental steps and accumulate the error to ideal line
  - Move one pixel up if error beyond a limit
  - Uses only integer arithmetic to ease computation
  - In each step:
    - 2 comparisons
    - 3 or 4 additions
- Let’s assume a simplified version first
  - We restrict 0 <= m <= 1
  - Work only in 1 positive octant
  - We have the 2 options
    - It may plot the point (x+1,y), or:
    - It may plot the point (x+1,y+1)

Rasterisation

- Bresenham‘s Algorithm
  - Define an error ϵ with each ordinate
  - ϵ is between +0,5 and -0,5
  - With each drawing step we increase ϵ by m
  - We draw (x+1,y) if y+ϵ+m < y+0,5 otherwise we draw (x+1,y+1)
  - The error is written back into ϵ
  - Depending on what is drawn the error can be
  - (y+ ϵ +m)-y or (y+ ϵ +m)-y+1
  - We don’t have multiplications in this approach anymore but still floating point values



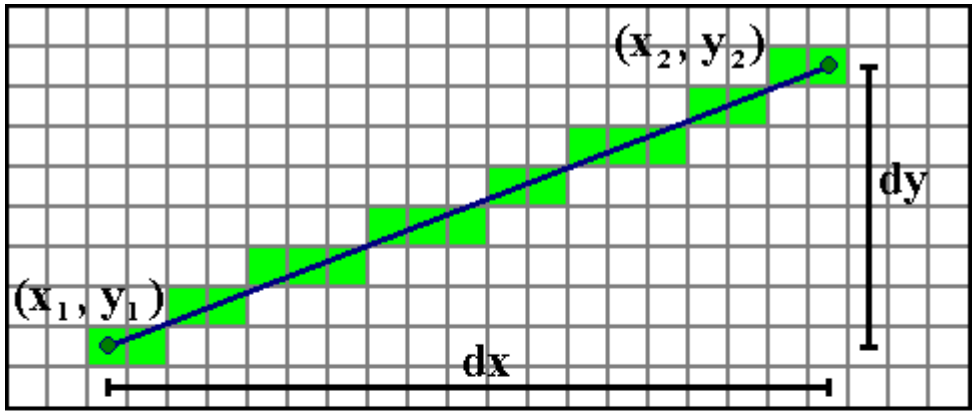
$\epsilon \leftarrow 0,\ y \leftarrow y_1$
for $x \leftarrow x_1$ to x_2 **do**
  Plot point at (x, y)
  if $\epsilon + m < 0.5$ **then**
    $\epsilon \leftarrow \epsilon + m$
  else
    $y \leftarrow y + 1,\ \epsilon \leftarrow \epsilon + m - 1$
  end if
end for

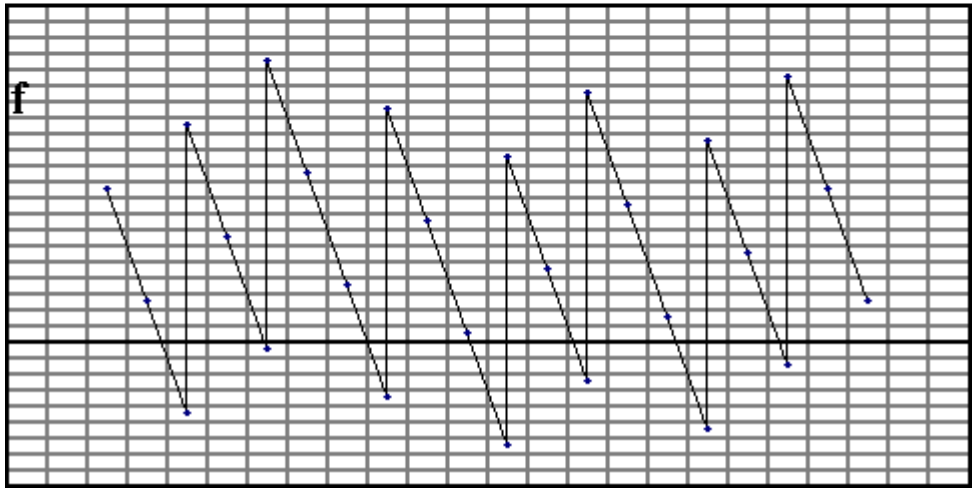
Rasterisation

- Bresenham‘s Algorithm
  - To avoid floating point usage, we simply double the values of the control variable

$\epsilon' \leftarrow 0,\ y \leftarrow y_1$
for $x \leftarrow x_1$ to x_2 **do**
  Plot point at (x, y)
  if $2(\epsilon' + \Delta y) < \Delta x$ **then**
    $\epsilon' \leftarrow \epsilon' + \Delta y$
  else
    $y \leftarrow y + 1,\ \epsilon' \leftarrow \epsilon' + \Delta y - \Delta x$
  end if
end for

  - Multiplication by 2 can be implemented by a bitshift



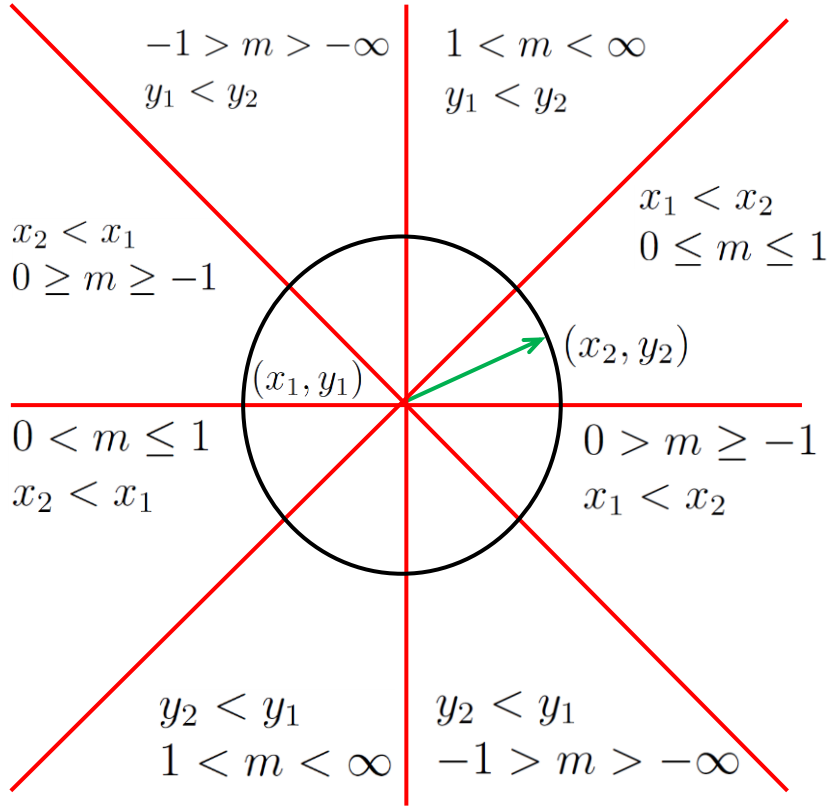


https://upload.wikimedia.org/wikipedia/commons/6/66/BresenhamLine2.png

Rasterisation

- Bresenham‘s Algorithm
  - So far it works only for the 1st octant
  - Some octants (diagonally opposed ones) can be easily handled by exchanging the endpoints of the lines

- Remaining problems
  - Drawing circles
  - Hard edges at lines



Anti-Aliasing

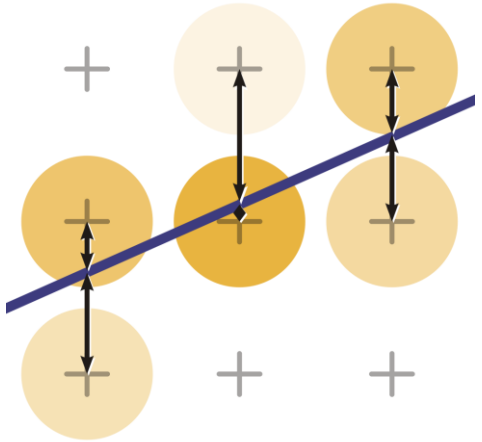
- Anti-aliasing
  - Moiré effects could appear through sampling during rasterisation
  - Idea to improve visual quality and remove sampling artefacts
  - Alternative or additional sampling has to be applied
  - Variety of sampling approaches exist
  - Anti-aliasing is typically set by the graphics board, but can also be performed manually in software
  - Multiple sub-sequential anti-aliasing stages can be performed
  - If amount of stages is too high the performance can drop drastically

https://upload.wikimedia.org/wikipedia/commons/3/31/Moire_pattern_of_bricks.jpg

https://upload.wikimedia.org/wikipedia/commons/f/fb/Moire_pattern_of_bricks_small.jpg

Anti-Aliasing

- Wu‘s Anti-aliasing Approach
  - Idea to have both pixels, on top and below the ideal line coloured proportional to the vertical distance of the ideal line
  - The further the pixel is away from the ideal line the more transparent is its colour
  - Exception is begin and endpoint of the line
  - Besides begin and end point the line strength is always two pixels



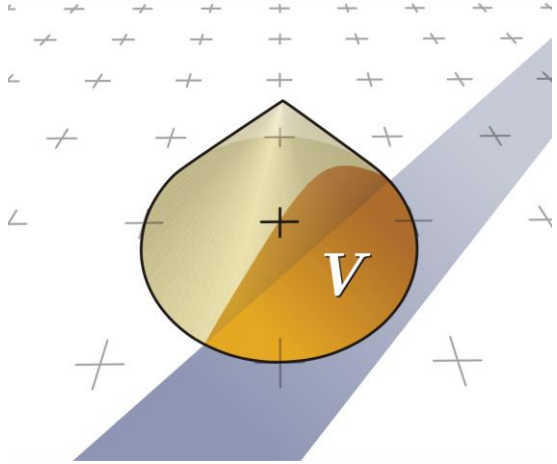
https://upload.wikimedia.org/wikipedia/commons/8/8a/Wu_line_drawing.svg

https://upload.wikimedia.org/wikipedia/commons/2/27/XiaolinWuLine.png

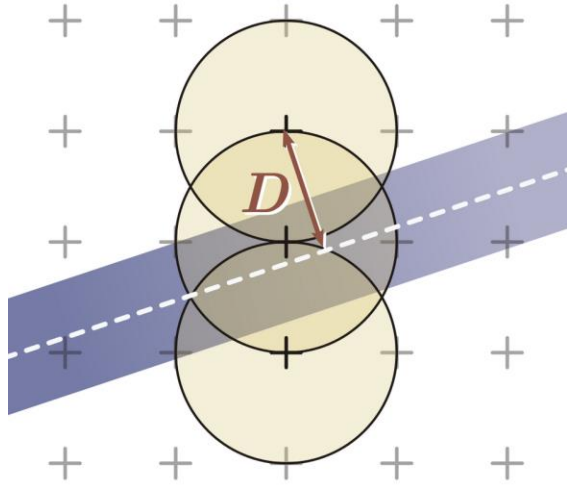
https://upload.wikimedia.org/wikipedia/commons/9/95/LineXiaolinWu.gif

Anti-Aliasing

- Gupta-Sproull Anti-aliasing
  - Use of a cone which is mapped on top of each pixel related to a line with the radius of one pixel
  - The resulting pixel colour is proportional to the cone volume covering the line which is assumed to have a thickness of one pixel as well
  - Calculation is approximated with the help of look-up tables
  - Length of a vector D, perpendicular to the line pointing to the pixel is also approximated and used for the lookup in the table



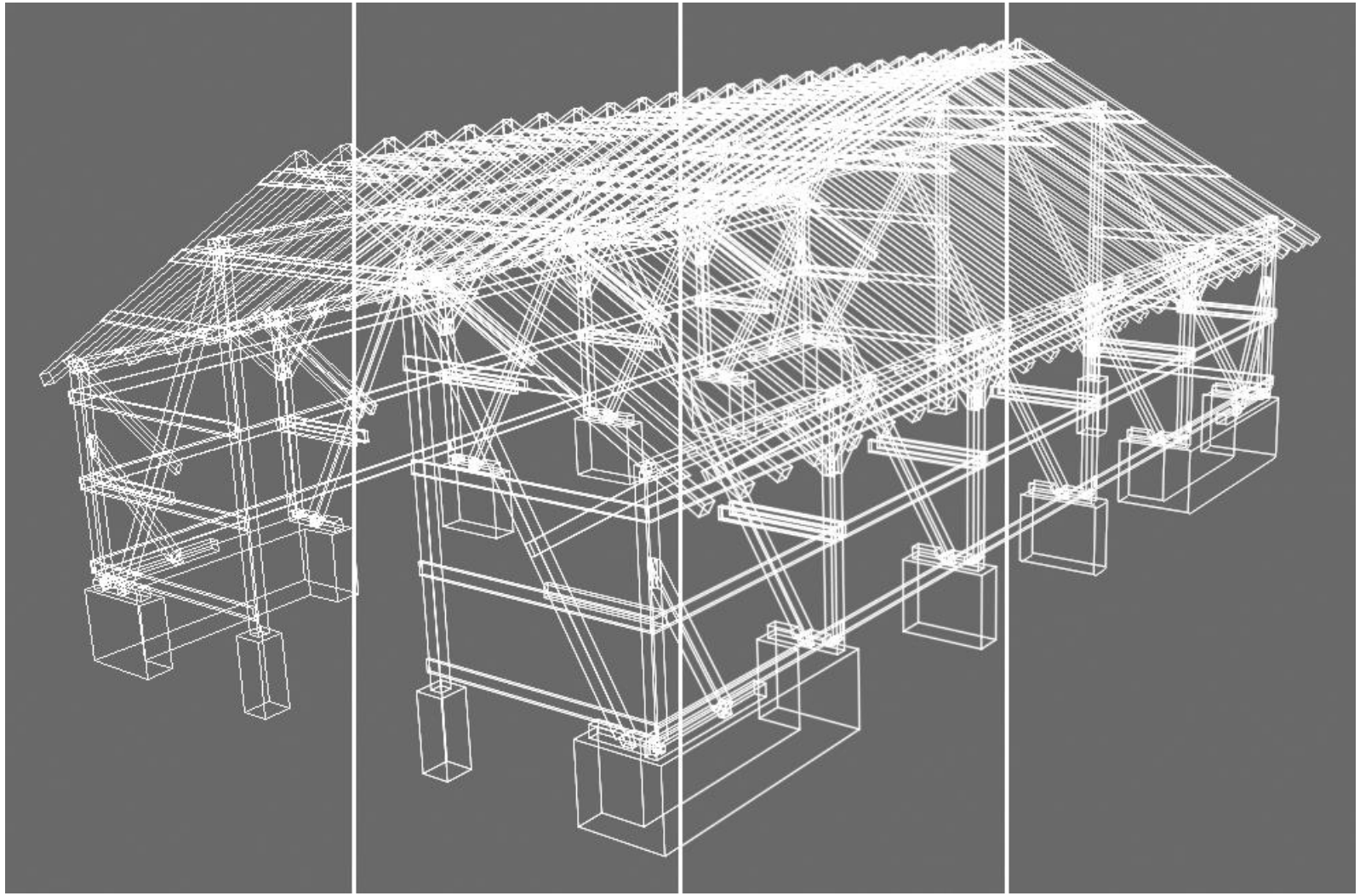
https://upload.wikimedia.org/wikipedia/commons/2/20/Gupta-Sproull_pixel_selection.svg



https://upload.wikimedia.org/wikipedia/commons/4/4a/Gupta-Sproull_filter_kernel.svg

Anti-Aliasing

- Comparison of 3 different anti-aliasing approaches
  - Left-most part shows no anti-aliasing (simple Bresenham)
  - Second from left Gupta-Sproull
  - Second from right unweighted surface sampling
  - Rightmost Wu
- More blurry but less edgy



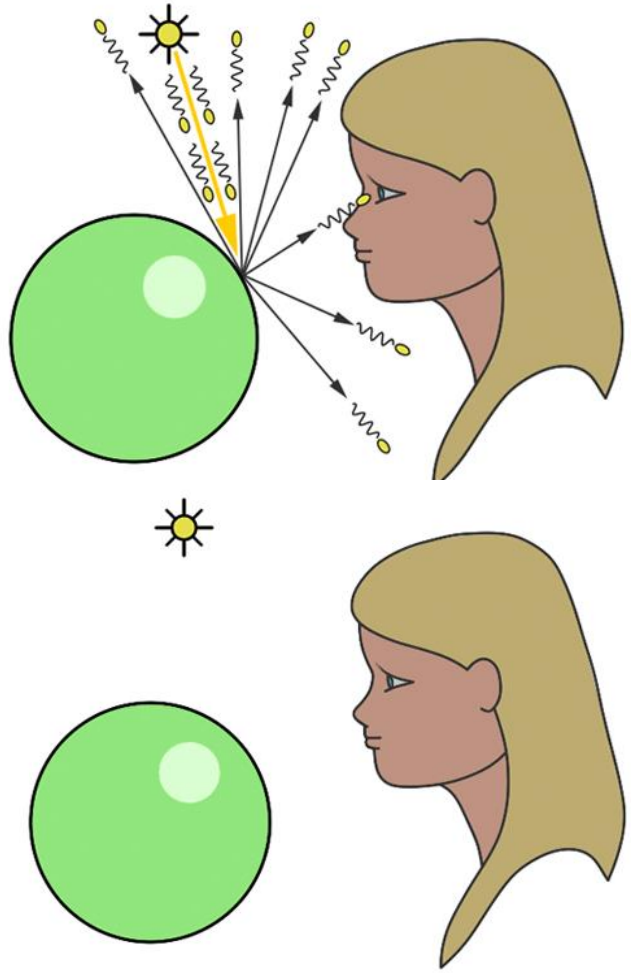
https://upload.wikimedia.org/wikipedia/commons/c/c3/Scheune_-_Antialiasing.png

Local Illumination

- So far we only looked at **local illumination** models
  - The colour of the pixel results from computations considering surface and light sources and maybe additional texture maps
- No indirect lighting (e.g. light reflected from other surfaces) has been considered so far
- Occluded surfaces are illuminated as well
- Many effects have been used to generate the illusion of a correct graphical representation of a real world object (e.g. reflection maps, shadow maps)

Raytracing

- General Idea
  - In the real environment rays of light emitted by multiple light sources hit surfaces, which again reflect the light and act as light sources themselves
  - Certain wavelengths are reflected depending on the material properties of the surface that is hit
  - At some point the reflected rays hit the eyes and we perceive the light
  - In raytracing we go the opposite way, by casting rays from the eye point in the scene
  - So far we only looked at the direct influence from the light sources affecting the hit point on the surface
  - In raytracing we do not stop at this calculation but iterate further



https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/raytracing-algorithm-in-a-nutshell

Raytracing

- Raytracing is the process of determining the visibility of surfaces by tracing imaginary rays of light from the viewer's eye (centre of projection) to objects in the scene

- If the cast ray stops after a hit on a surface we speak of ray casting, typically the raytracing algorithm is recursive
- The simple ray casting variant determines the surface colour based on the reflection models we have seen so far
- With raytracing we bounce the ray off from the hit surface to the next surface

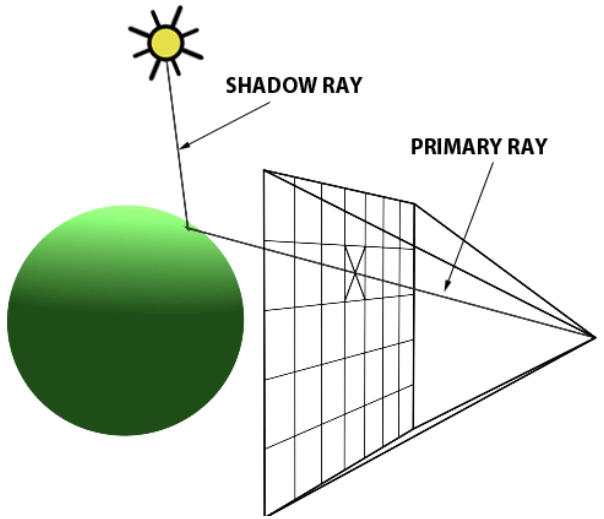
Raytracing

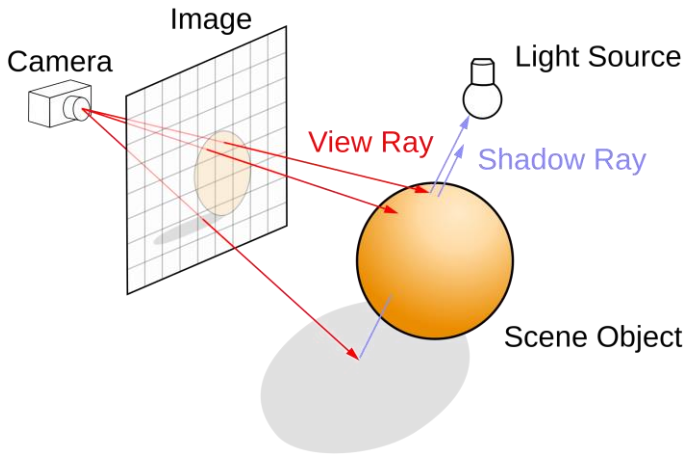
- Effects which can be easily implemented in raytracing
  - Reflections and refraction
  - Shadows
- Render complexity is determined by number of bounces and display resolution
  - Complexity of the scene plays a minor role
- Generates realistic looking images through approximated lighting computations

https://upload.wikimedia.org/wikipedia/commons/e/ec/Glasses_800_edit.png

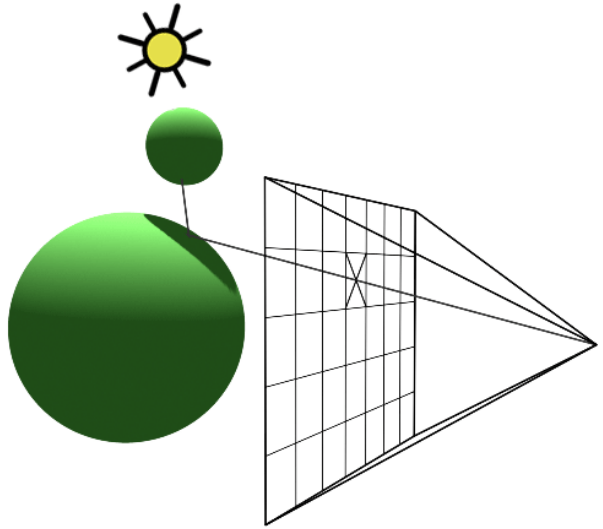
Raytracing

- Shadows
  - To calculate shadows we cast an additional ray from the hit surface to the light source
  - If we intersect another object before the light source we are in the shadow of that object
  - As a result the direct contribution from that light source is ignored
  - This is repeated for all light sources in the scene





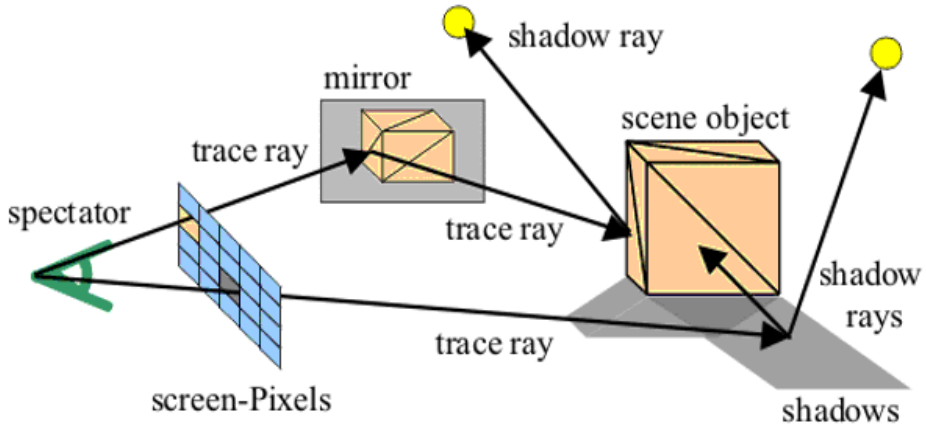
https://en.wikipedia.org/wiki/Ray_tracing_(graphics)#/media/File:Ray_trace_diagram.svg



https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/implementing-the-raytracing-algorithm

Raytracing

- Recursive raytracing in a nutshell
  - Ray from viewpoint through each screen pixel is followed
  - When it hits a surface the colour of the surface is determined by the contribution from the light sources
  - Shadow can be calculated with shadow rays
  - Ray can bounce off from the surface based on the angle it hit the surface
  - Colour of the hit point of the reflected ray on the second surface hit can be calculated as before
  - The calculated colour contributes to the first hit point and so on

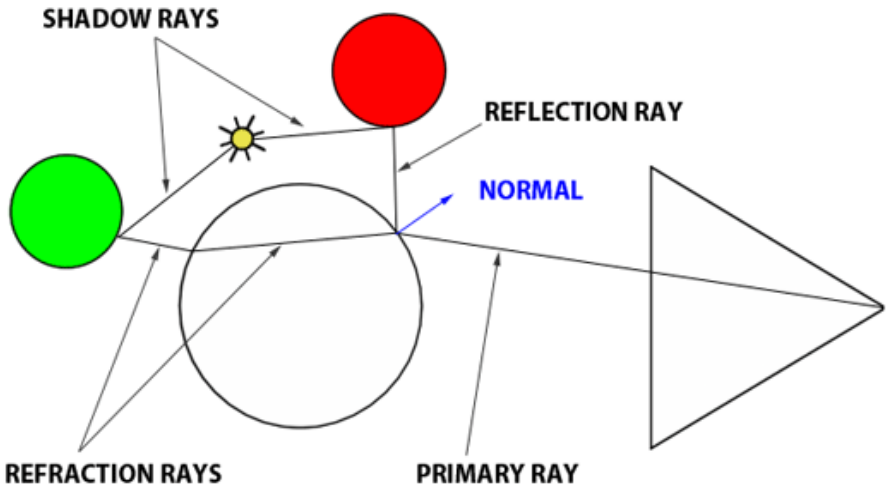


  - What about fully reflective surfaces and transparent surfaces?

Raytracing

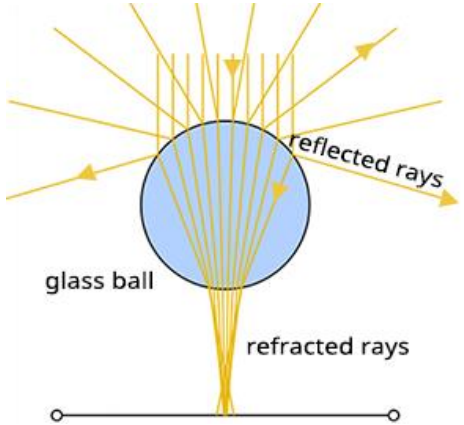
- Refraction and reflection
  - Both effects are considered in raytracing
  - For reflection we can calculate the reflection ray and continue from there on
  - We use the same equation as with the Phong reflection model to calculate $\vec{R}$

$$\vec{R} = 2(\vec{L} \cdot \vec{N})\vec{N} - \vec{L}$$



https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/adding-reflection-and-refraction

- Refraction takes place when light hits a surface between two transparent media (e.g. air-water, air-glass, glass-water)



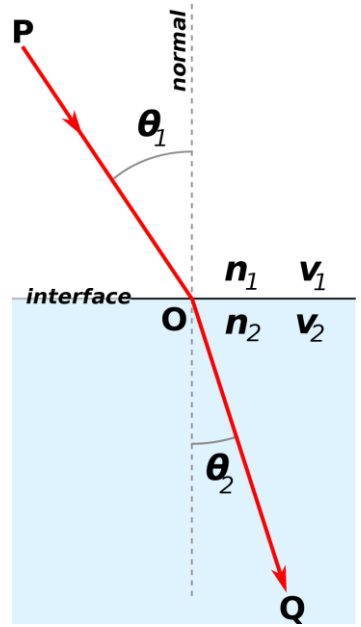
https://www.scratchapixel.com/images/upload/introduction-to-ray-tracing/glassball.png

Raytracing

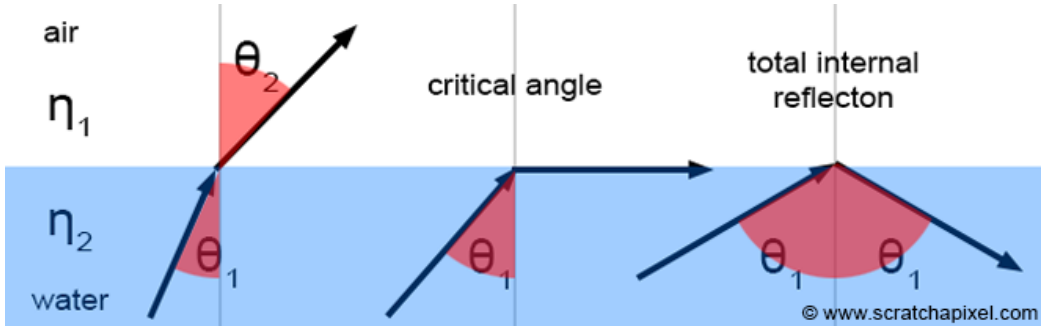
- The rays which hit a refractive surface change their direction
- The new direction depends on two factors
  - The angle of incidence
  - The refractive index of the new medium
- Snell's Law
  - For a given pair of media, the ratio of the sines of the angle of incidence θ_1 and angle of refraction θ_2 is equivalent to the opposite ratio of the indices of refraction

$$\frac{\sin\theta_2}{\sin\theta_1} = \frac{n_1}{n_2} = n_r$$

- Total Internal Reflection (TIR)
  - If angle is over critical value full reflection takes place



https://en.wikipedia.org/wiki/Snell%27s_law#/media/File:Snells_law2.svg

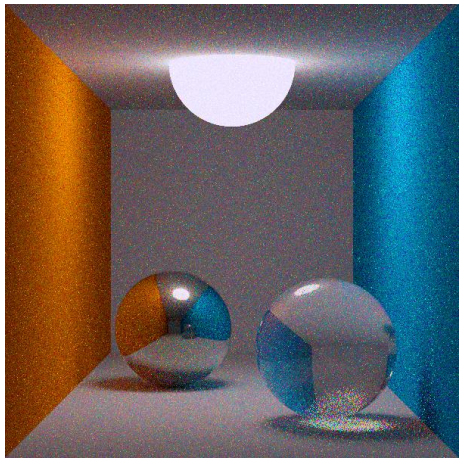


https://www.scratchapixel.com/images/upload/shading-intro/shad-refraction9.png?

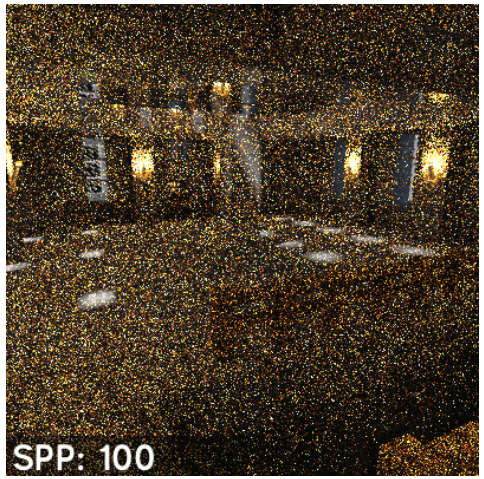
https://blogs.nvidia.com/wp-content/uploads/2018/03/what-is-ray-tracing-explaination-1280x725.jpg

Path Tracing and Light Raytracing

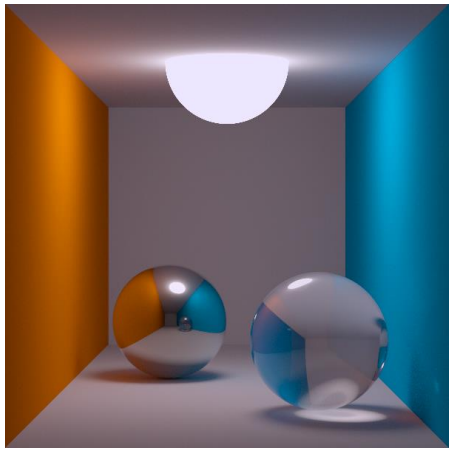
- Path tracing is an alternative to raytracing where light rays are still cast in the scene
- A ray hitting a surface is either reflected, refracted or absorbed
  - When reflected or refracted at least one additional random ray is created
- Opposed to raytracing many rays are cast through a pixel
- The more rays are cast inside the scene the closer we come to a realistic representation of the scene, quality is defined by number of samples taken
- Monte Carlo integration to solve the rendering equation by approximation
- Fun: Disney's Practical Guide to Path Tracing
  - https://www.youtube.com/watch?v=frLwRLS_ZR0



https://de.wikipedia.org/wiki/Path_Tracing#/media/File:Box_-_Path_Tracing_Low.png



https://chunky.llbit.se/path_tracing.html

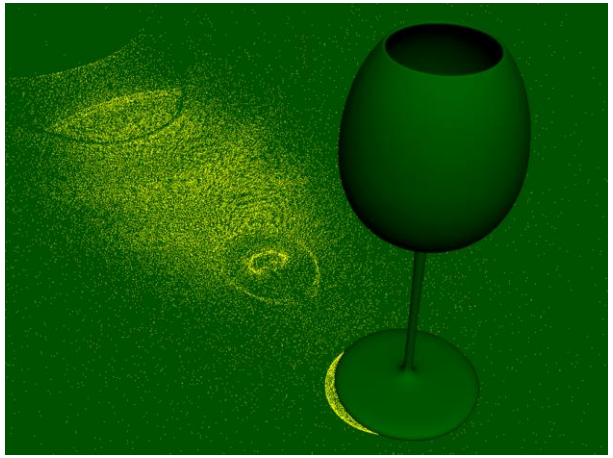


https://upload.wikimedia.org/wikipedia/commons/0/0e/Box_-_Path_Tracing_High.png

Photon Mapping

- Two-pass rendering approach for global illumination
- Light is emitted from the light sources in form of photons
- These photons can be reflected, refracted, scattered, absorbed
- When a diffuse surface is hit the photon is stored inside a three-dimensional surface structure, the photon map
- Data from the photon map is used for colour calculation additionally a caustic map is calculated

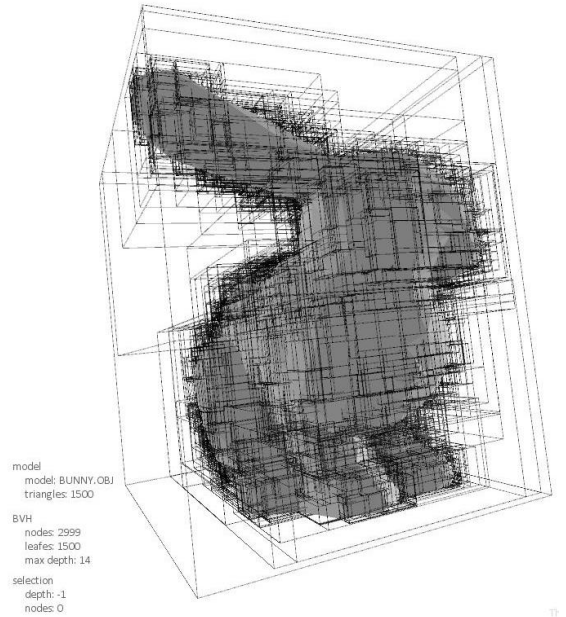
https://de.wikipedia.org/wiki/Photon_Mapping#/media/File:Glas-2000-enery.jpg



https://de.wikipedia.org/wiki/Photon_Mapping#/media/File:Caustics.png

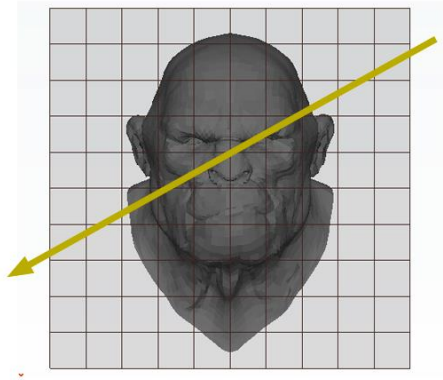
Acceleration Structures

- Intersection calculation is expensive
- Key aspect for raytracing are acceleration structures to reduce the amount of required ray calculations and intersections
  - Use of bounding boxes
  - Use bounding volume hierarchy (BVH)
  - Spatial trees, Octrees, BSP
  - Grid structures, only intersection tests with grid cells at first



https://tobiasbu.github.io/img/portfolio/raytracer_meshbvh.jpg

- Could be parallelised well, a single core can in theory handle a single pixel
- Problem with scene distribution and communication overhead between cores



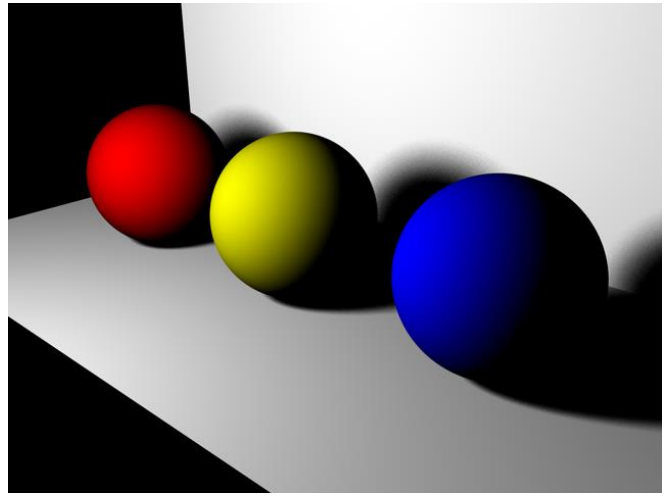
https://www.scratchapixel.com/images/upload/acceleration-structure/grid1.gif?

https://www.youtube.com/watch?time_continue=3&v=vY0W3MkZFs4

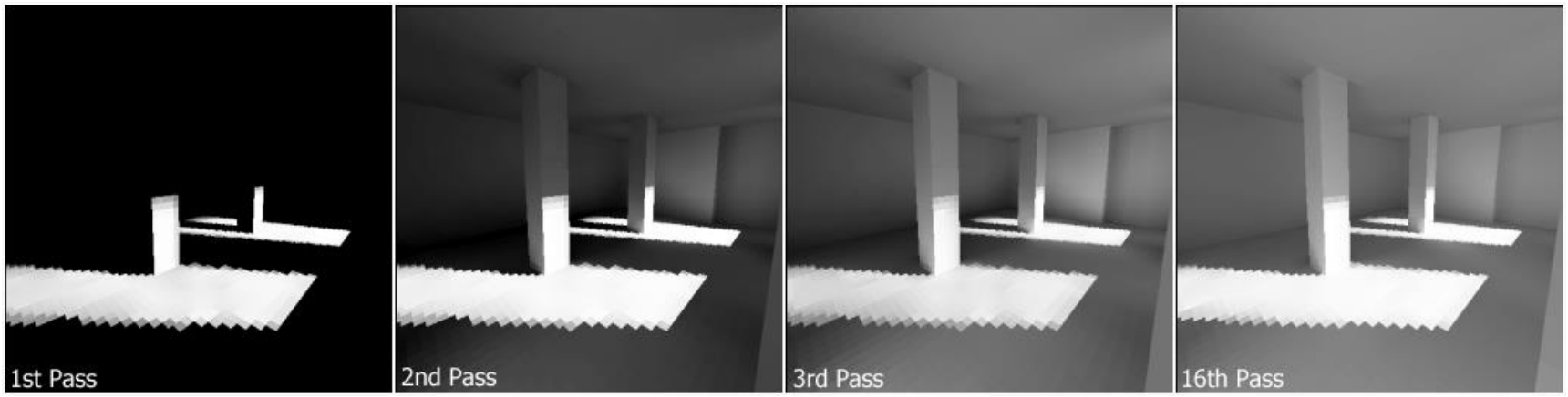

https://www.youtube.com/watch?v=hzk1IEFR1yc

Radiosity

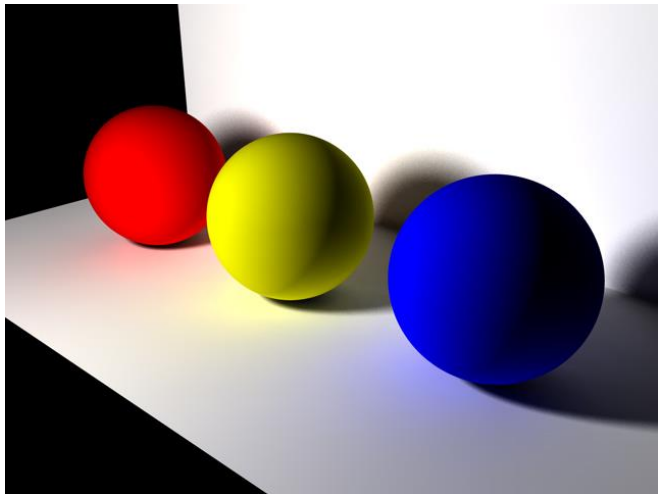
- Another global illumination approach is Radiosity
  - While raytracing is focused on optical geometry Radiosity is a finite elements based approach to solve the rendering equation
  - Based on the conservation-of-energy principle, all light that is not absorbed has to be reflected
  - Multiple iterations are required to generate an image



https://upload.wikimedia.org/wikipedia/commons/f/f2/Radiosity-no.jpg



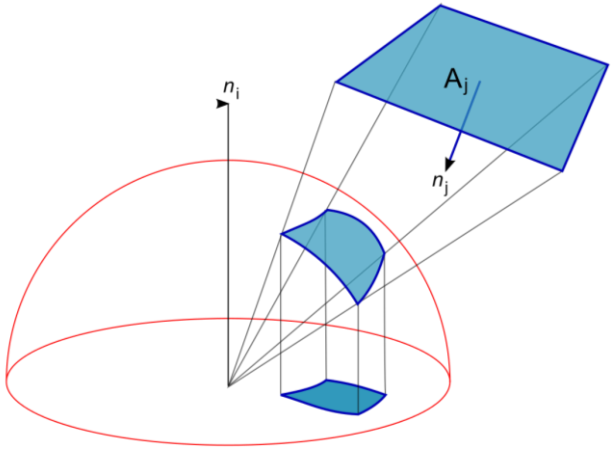
https://upload.wikimedia.org/wikipedia/commons/f/f2/Radiosity_Progress.png



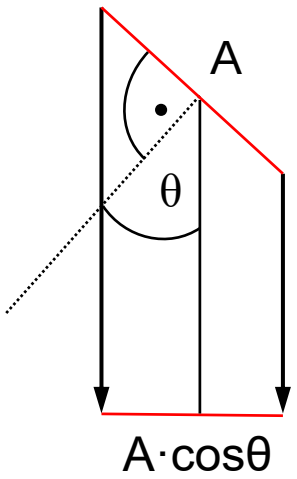
https://upload.wikimedia.org/wikipedia/commons/4/43/Radiosity-yes.jpg

Radiosity

- Approach
  - Divide surfaces into patches (elements)
  - Model light transfer between patches as system of linear equations
  - Reflection and emission are diffuse
  - All surface are ideal diffuse surfaces
  - So radiance is independent of direction
  - Form factor F_{ij}:
    - Fraction of light leaving element i arriving at element j
    - Depends on
      - Shape of patches i and j
      - Relative orientation of both patches
      - Distance between patches
      - Occlusion by other patches

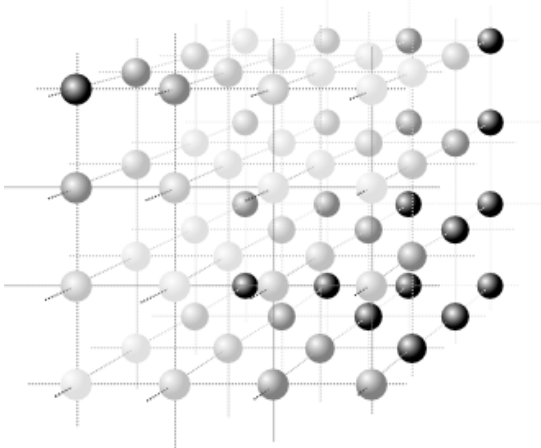


https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Nusselt_analog.svg/1280px-Nusselt_analog.svg.png

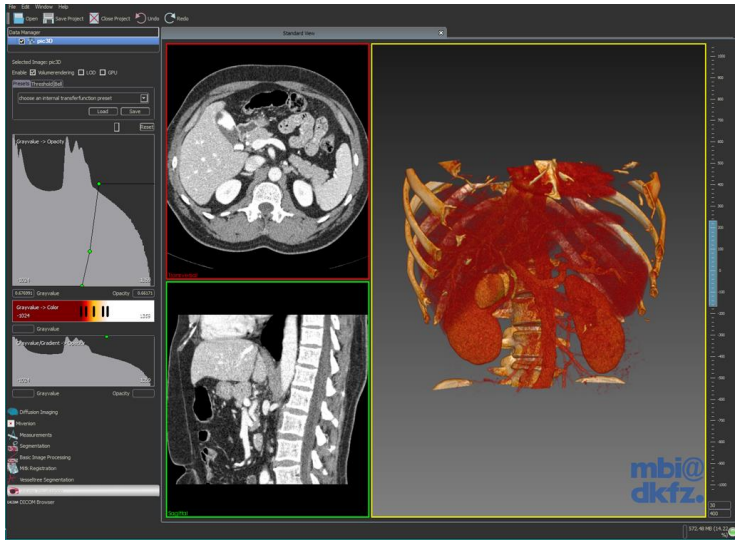


Volume Rendering

- Raycasting through a volume
- Voxel representation
  - Space is separated in to voxels (volume element comparable to 2d pixel but in 3d space)
  - A voxel is a point an has no real dimension opposed to a pixel
- Small volumes can be calculated in real time, large volumes are typically computed offline
- Often conversion in a surface with a defined density value is desired (isosurface)
  - With the surface traditional triangle rendering can be performed
- Transfer functions define the opacity and colour of a rendered volume



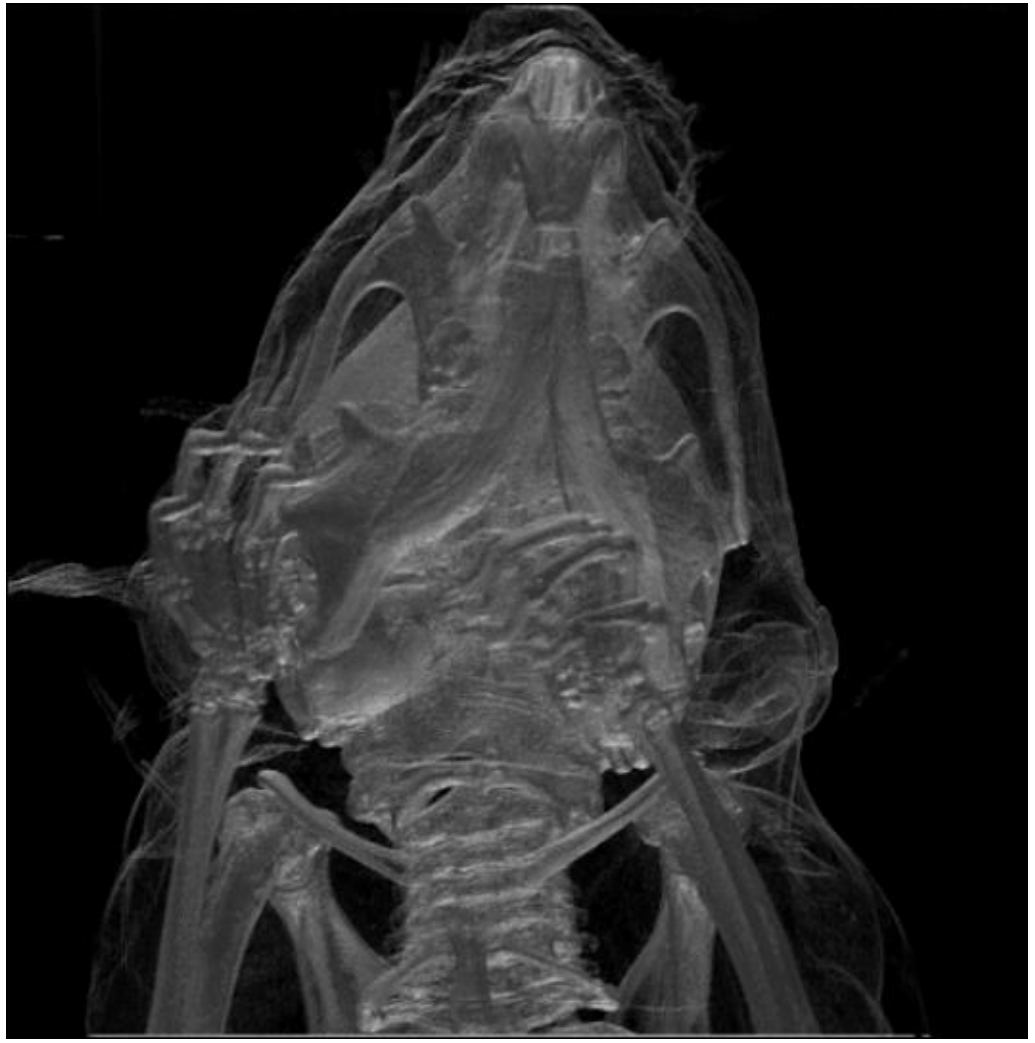
https://upload.wikimedia.org/wikipedia/commons/b/b4/Voxelgitter.png



http://docs.mitk.org/3m3-1.1/vroverview.png

Volume Rendering

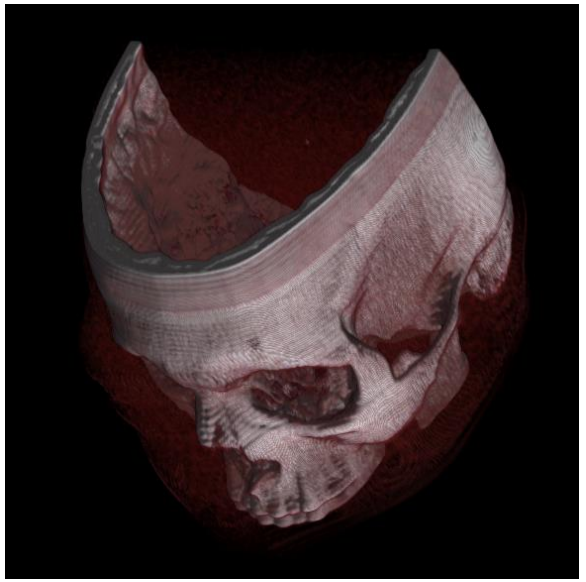
- Algorithms to render Voxel data structures
- Separated into 4 steps
  - Classification
    - Definition of the voxel values inside the grid (e.g. x-ray density in CT) through a transfer function
  - Shading
    - Often implemented with Phong reflection
  - Interpolation
    - Voxels are defined as points this interpolation between voxels becomes necessary
  - Compositing
    - Summing up of the calculated voxel values



https://de.wikipedia.org/wiki/Volumengrafik#/media/File:VolRenderShearWarp.gif

Volume Rendering

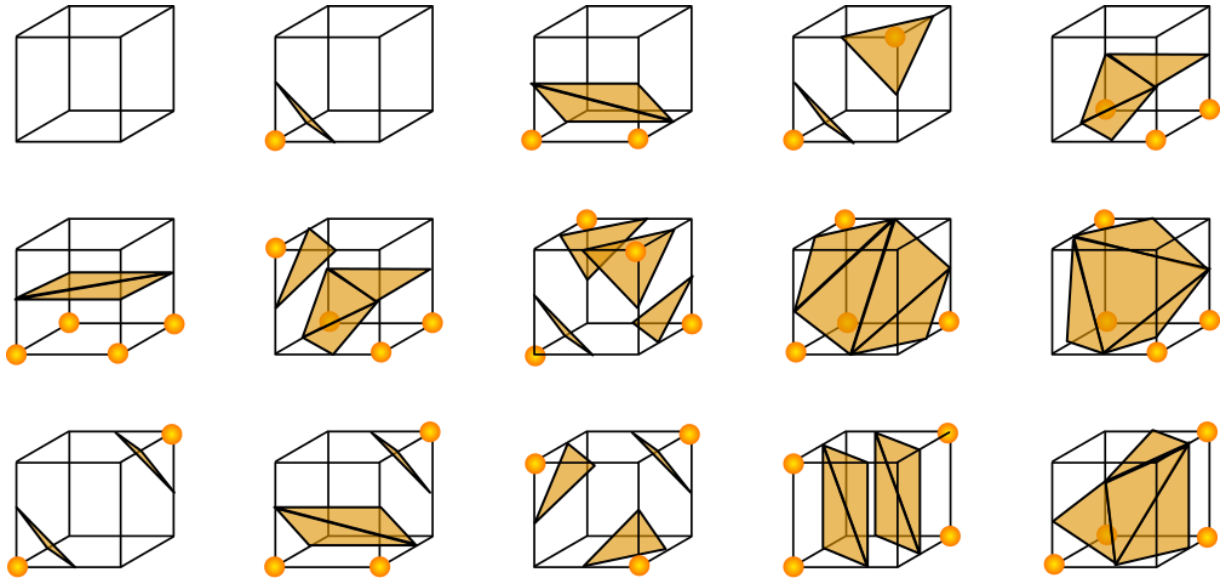
- Volume rendering is computationally intense a variety of optimisation techniques have been developed
  - Volumes have often large empty areas, space leaping can be performed to skip empty cells
  - Use of acceleration structures like Octrees and BSP
  - Early-ray termination - faraway regions do not contribute in case accumulated opacity is to high, we can stop the traversal if contribution of the sample becomes irrelevant
  - Homogeneity-acceleration – we can approximate homogeneous regions with fewer sample points
- Different techniques for volume rendering exist
  - Volume ray casting
  - Splatting
  - Shear warp
  - Texture-based volume rendering



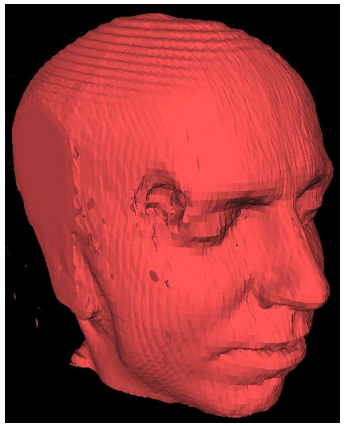
https://upload.wikimedia.org/wikipedia/commons/d/d2/CTSkullImage.png

Generating Isosurfaces

- To generate isosurfaces (polygonal surfaces, built of a constant value in a voxel grid) several algorithms exist
- Advantage of isosurfaces is the fast rendering
- Can be optimised with octree
- Marching cube algorithm
  - Cube is moved with the offset of 0,5 voxel width through the volume
  - Triangle Lookup Table (TLT) provides information which triangles are to be generated, if voxel with desired density value is intersected
  - Binary encoded indexation of TLT based on the hit corners
  - In general 256 different combinations are thinkable



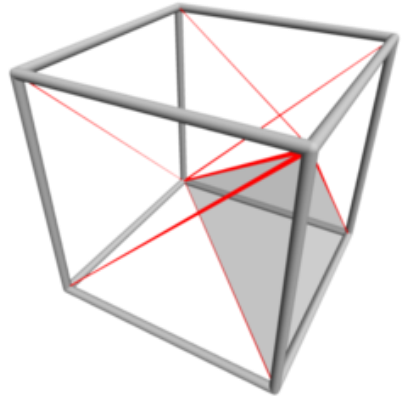
https://de.wikipedia.org/wiki/Datei:MarchingCubes.svg



https://upload.wikimedia.org/wikipedia/commons/6/63/Marchingcubes-head.png

Generating Isosurfaces

- Marching Tetrahedra
  - Alternative because marching cube was patented for 20 years
  - Cube is split into 6 irregular tetrahedra which all share one diagonal in the cube
  - Configurations
    - No intersection
    - Intersection in one triangle
    - Intersection in two adjacent triangles
  - Considers 19 edge intersections



https://en.wikipedia.org/wiki/Marching_tetrahedra#/media/File:Marching_tetrahedrons.png


https://vimeo.com/158172041

Bibliography

- Bresenham, J. E. Algorithm for computer control of a digital plotter IBM Systems Journal, IBM, 1965, 4, 25-30
- Wu, X. An efficient antialiasing technique ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1991, 25, 143-152
- Gupta, S. & Sproull, R. F. Filtering edges for gray-scale displays Proceedings of the 8th annual conference on Computer graphics and interactive techniques - SIGGRAPH '81, ACM Press, 1981
- Appel, A. Some techniques for shading machine renderings of solids Proceedings of the April 30-May 2, 1968, spring joint computer conference on - AFIPS '68 (Spring), ACM Press, 1968
- Whitted, T. An improved illumination model for shaded display Communications of the ACM, Association for Computing Machinery (ACM), 1980, 23, 343-349
- Glassner, A. S. An Introduction to Ray Tracing ACADEMIC PR INC, 1989
- Jensen, H. W. Realistic Image Synthesis Using Photon Mapping, 2nd Edition A K PETERS LTD (MA), 2001

Bibliography

- Immel, D. S.; Cohen, M. F. & Greenberg, D. P. A radiosity method for non-diffuse environments Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘86, ACM Press, 1986
- Lorensen, W. E. & Cline, H. E. Marching cubes: A high resolution 3D surface construction algorithm Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87, ACM Press, 1987
- Akio Doi, Akio Koide. "An Efficient Method of Triangulating Equi-Valued Surfaces by Using Tetrahedral Cells." IEICE Transactions of Information and Systems, Vol.E74-D No. 1, 1991

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Computer Graphics

Animation, Particle Systems, Swarming, Steering

FH-Prof. Dr. techn. Dipl.-Inf. (FH) Christoph Anthes, MSc

Professor of Augmented and Virtual Reality

Overview

- Animation
- Skeletal Animation
  - Free Form Deformation
- Particle Systems
- Swarming and Flocking
- Steering Behaviours
  - Vehicle Model
  - Behaviours

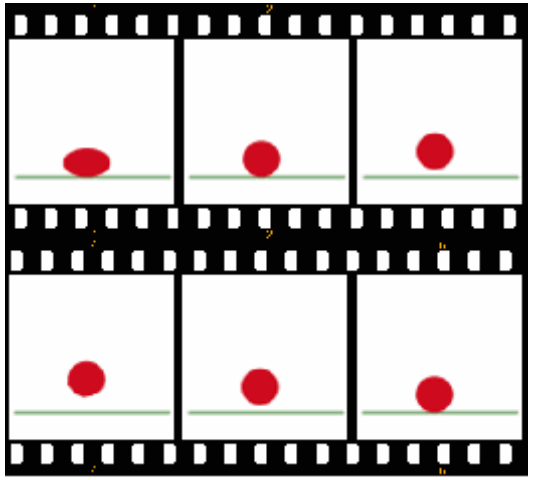
Animation

- To perceive a fluent animation we need at least 24 frames per second (FPS)
  - In European TVs used to be 24 FPS (PAL/SECAM)
  - US TVs displayed at 30 FPS
  - Some cinema productions tried to go for high frame rates (HFR) 48 FPS (e.g. The Hobbit)

- Dynamic graphical effects
  - Can increase perceived realism of the scene
  - Increase place and plausibility illusion in Virtual Reality
  - Can be used for the simulation of complex systems
  - Computer generated artworks

Animation

- Typically animation takes place through scripting
  - Example keyframe animation
  - Movement of components from A to C through point B
  - Interpolation is required
  - Most simple interpolation is linear interpolation which quite often produces bad quality
  - Other interpolations possible, cosine, quadratic, etc.



https://upload.wikimedia.org/wikipedia/commons/e/ee/Animexample3edit.png

- Problems with scripted animation are the degrees of freedom
  - Cube has 6DOF in three-dimensional space
  - Simple model might have 20 DOF
  - How many DOF has human skin, a flock of birds, a flame?

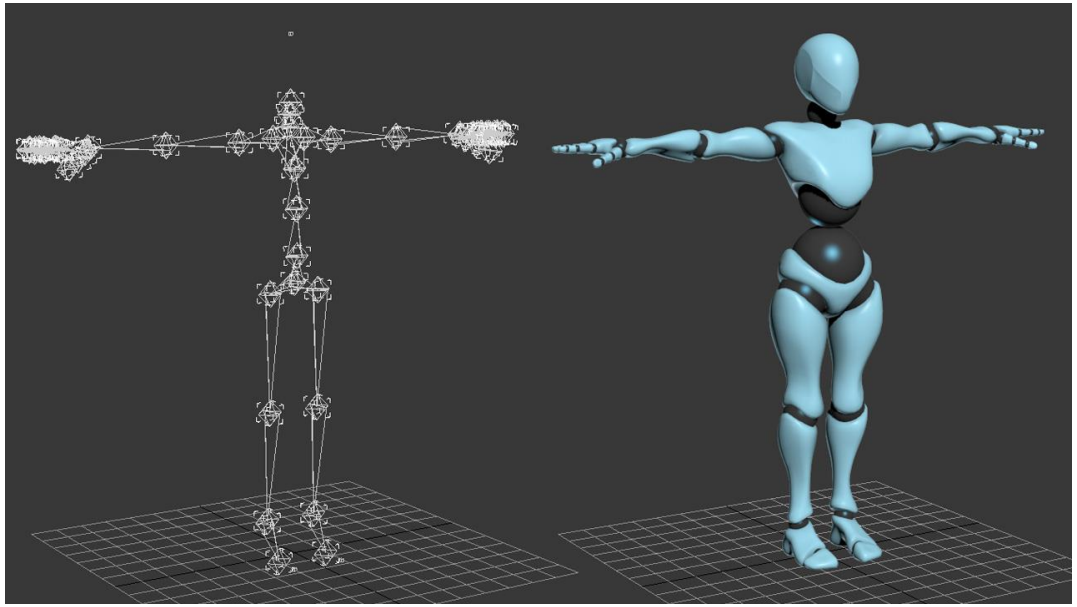
https://upload.wikimedia.org/wikipedia/commons/a/a4/Animexample.gif

- Variety of algorithms available for animation
  - Focus here on skeletal animation, particle systems swarming and steering algorithms

https://www.youtube.com/watch?v=QfRAfsK5cvU

Skeletal Animation

- Idea to animate characters
  - Use of a skeleton consisting of bones and a mesh representing the character surface
  - Vertices of the mesh are associated with bones
  - Process of association is called rigging or mesh skinning
  - When the bone is moved the vertices and thus the mesh is moved relative to the bone
  - Advantage: only a comparable small set of bones has to be transformed when the whole mesh should be animated
  - Potential problem - joints
    - Where are the vertices at a joint to be transformed?
    - Solution: use of weighing tables associating single vertices to multiple joints



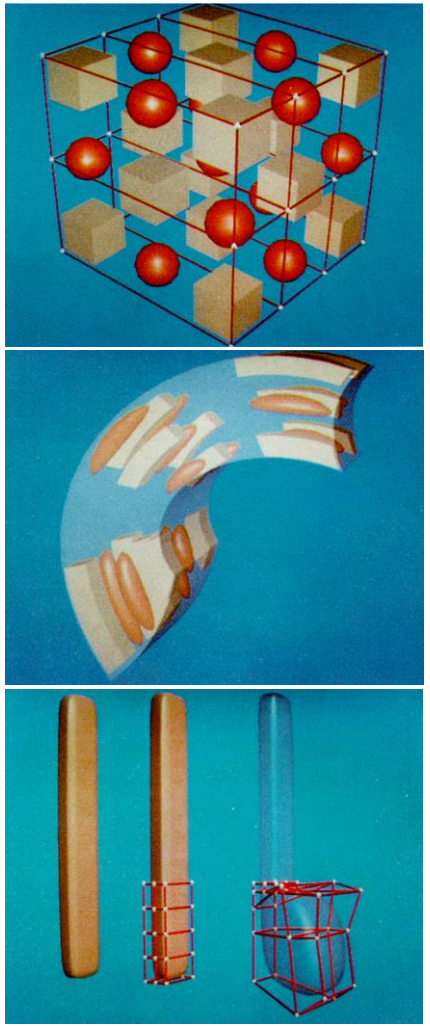
http://www.downloads.redway3d.com/downloads/public/documentation/wf_SkeletalAnimation01.jpg

Skeletal Animation

- Animation of characters
  - Complex motion can be scripted and transferred to other models, only the rigging has to be recomputed, the defined animation can stay the same
  - Instead of scripting rigid body dynamics can be used (Ragdoll modelling)
  - Use of morphs and blend shapes additionally deforming the mesh to increase realism
  - Can be computationally intense, due to the amount of vertices to be individually transformed each frame
- Virtual reality specifics
  - In virtual reality input of position tracking often only very coarse (amount of sensors is often limited – common options 2, 10, or 16 sensors)
  - Motion of the user should be mapped on avatar motion (self-representation or remote collaboration)
  - Animation via inverse kinematics possible

Free Form Deformation

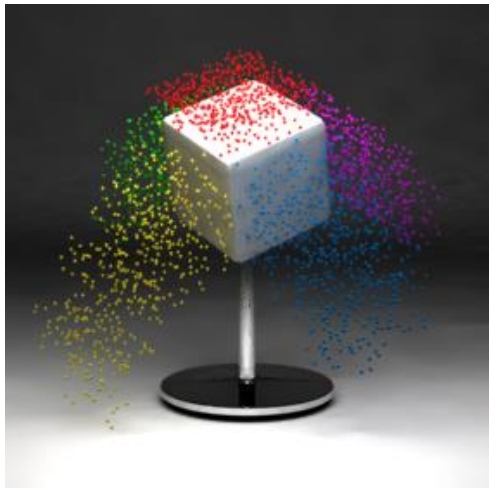
- Free From Deformations (FFD) are mechanisms to deform a whole group of objects or triangles
- The idea behind FFD is to place the polygons with their vertices into a grid
- The deformation or transformation takes place on the individual grid point thus deforming the cells of the grid
- Individual vertices inside the cells are deformed proportionally to the deformation of the cell respectively the vertices
- Can be applied locally on parts of the mesh or globally on the whole mesh
- Current applications in 3d modelling, character animation, facial animation



Sederberg, T. W. & Parry, S. R. Free-form deformation of solid geometric models *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH '86, ACM Press,* 1986

Particle Systems

- Motivation
  - Visualisation of amorphous structures (no clear surface)
  - Modification of large objects through change of individual components
  - Volume visualisation
- Key attributes of particle systems
  - Clouds of many particles
  - Dynamic and non-deterministic
  - Way of three-dimensional modelling and animation
- Examples
  - Many natural phenomena like
  - Clouds, fire, smoke, water, fog, sand storms, explosions, fireworks, snow, rain
  - Extension plants (Reeves and Blau), motion blur (Sims)



https://upload.wikimedia.org/wikipedia/commons/1/1c/Particle_Emitter.jpg

https://upload.wikimedia.org/wikipedia/commons/4/44/Strand_Emitter.jpg

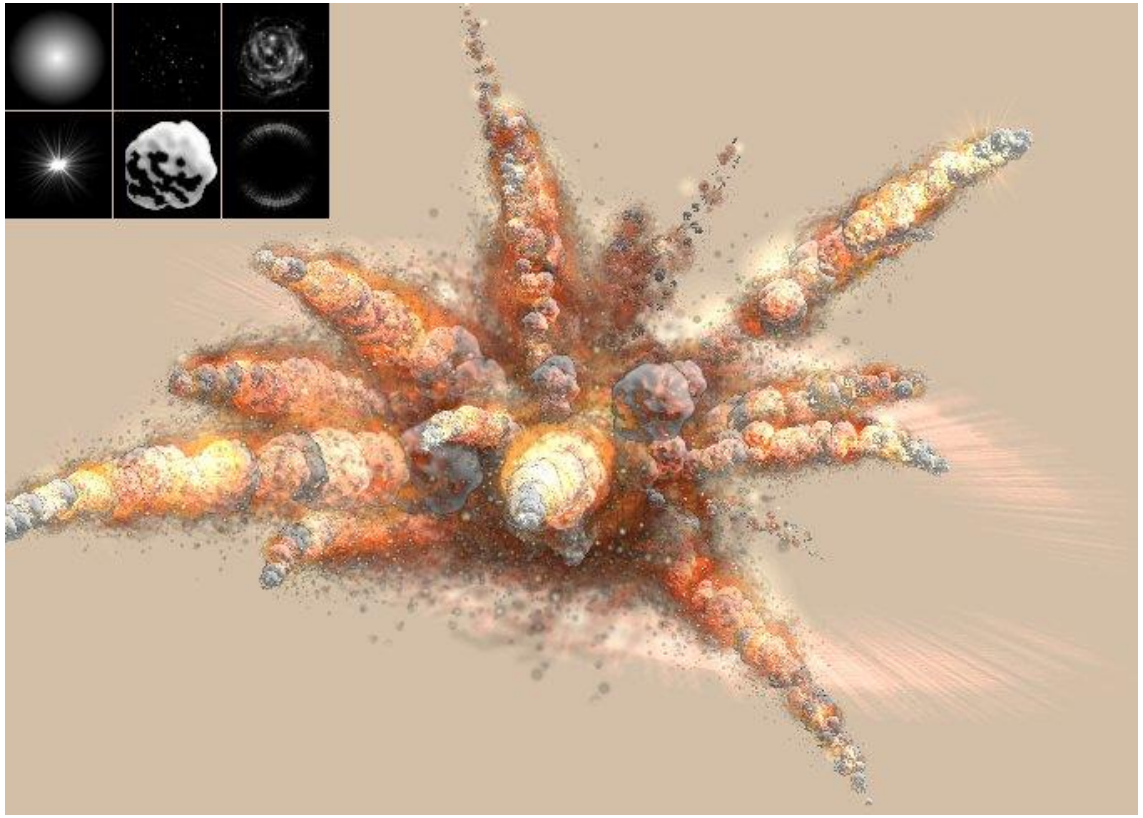
dissolving particle shapes

VEX Particle Count 10 - 32 Million

https://www.youtube.com/watch?v=AcPbn4w5Vvw

Particle Systems

- Structure
  - Particles – the individual elements to be displayed
  - System core – control over particles
- Particle attributes
  - Individual attributes of a single particle
  - Attributes depend on system and use case
  - Can be static or dynamic
  - Can be dependant on other attributes of the same particle
  - Generally important attributes
    - Position, speed, direction
    - Colour, transparency, size, shape
    - Life span



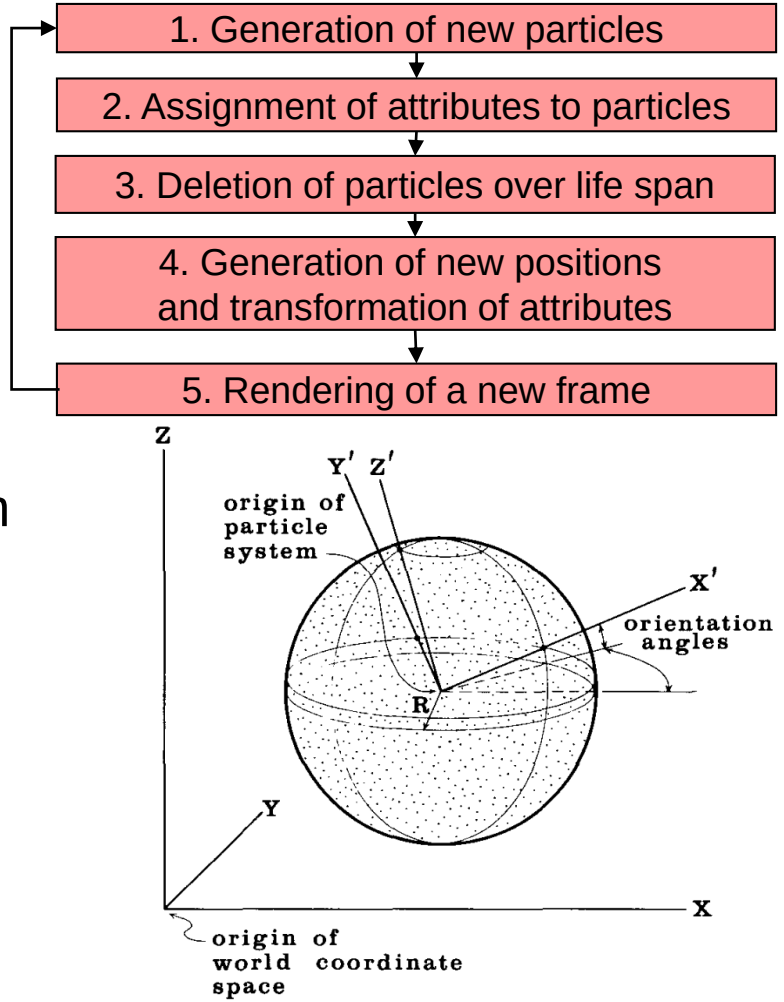
https://upload.wikimedia.org/wikipedia/commons/2/28/Pi-explosion.jpg

Particle Systems

- Particle attributes in detail - Attributes are valid only for fixed time (e.g. a frame)
  - **Position** – describes point in space where particle is located at a given time
  - **Speed and direction** – Acceleration and direction vector, can change dynamically (e.g. through collision, gravitation)
  - **Colour** – can change over time, (e.g. explosions typically start with white and fade into red)
  - **Transparency** – changes also over time with help of alpha channel
  - **Size** – can be modelled as points (size=0), when modelled as multidimensional structure size can be changed over time
  - **Life span** – time between particle generation (birth) and deletion (death)
  - **Shape (not required)** – geometry displayed at particle position (e.g. rectangle with billboard)
- Extensions
  - Age – reduction of age dependant calculations
  - Previous position – can be used for effect like motion blur
  - Rotation parameters – (orientation not necessarily required)

Particle Systems

- System core
  - Controls creation and destruction of particles
  - Controls particle attributes
  - Has a global coordinate system describing particle transformation
  - Procedural system separated into 5 phases
  - Multiple system cores can form a hierarchical particle system
- Phase 1 – Generation of new particles
  - The amount of particles to be generated is important
    - It has a strong influence on the density of the object
  - Two stochastic approaches are common
    - Total amount of particles can be set
    - Amount of particles depending on the image section



W. T. Reeves, “Particle Systems—a Technique for Modeling a Class of Fuzzy Objects,” *ACM Transactions on Graphics*, vol. 2, no. 2, pp. 91–108, Apr. 1983.

Particle Systems

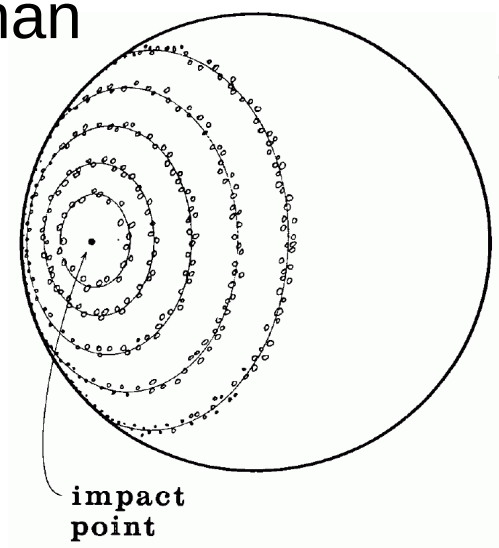
- Phase 1 – Generation of new particles
  - $\mathrm{MeanParts}_f$ – average amount of particles per frame
  - $\mathrm{VarParts}_f$ – variance
  - $\mathrm{rand}()$ – randomizer generating floating point values between -1 and 1
  - $\mathrm{NParts}_f = \mathrm{MeanParts}_f + \mathrm{rand}() * \mathrm{VarParts}_f$ – amount of generated particles
- Phase 2 – Assignment of attributes to the particles
  - Particles are created inside a defined initial system geometry and initialised
  - Typically the geometry also indicates a direction in which the particles should move (e.g. sphere (motion away from the centre), circle (motion away from a plane))
  - $\mathrm{InitialSpeed} = \mathrm{MeanSpeed} + \mathrm{rand}() * \mathrm{VarSpeed}$ – initial speed of a particle
  - Colour, size and transparency are determined with similar equations
  - Shape is typically fixed, but can also be changed dynamically
  - The life span of a particle is determined

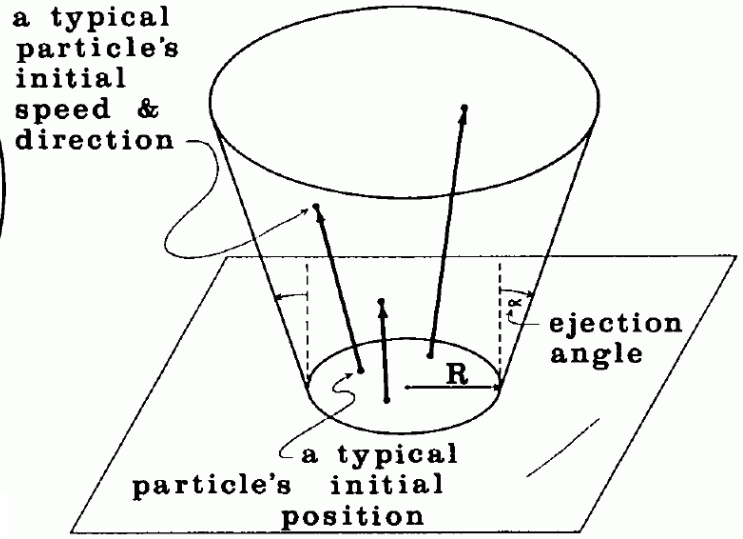
Particle Systems

- Phase 3 – Deletion of particles of which the life span is over
  - Life span initially set is decremented
  - Particles are removed if life span counter is 0, and when they are not visible anymore (e.g. transparency is 100%)
  - Particles as in terms of data structure should be reused
- Phase 4 – Generation of new positions and transformation of dynamic attributes
  - Particles move in space and change their colour, transparency, size and shape over time
  - Positional change is implemented by addition of the speed vector to the position vector
  - Additionally gravitation and acceleration can be considered
- Phase 5 – Rendering of a new frame
  - After all transformations and attribute changes are performed new particles can be displayed
  - Overlapping particles can for example be considered light emitting, they become brighter
  - General issues exist with shadows and collision

Particle Systems

- Hierarchical particle systems
  - Particles can be particle systems themselves
  - Data of the system core influences the system cores of the new particle systems
    - Clouds as top particle systems – are affected by influences like wind and terrain
    - Cloud elements are sub particle systems – monition of air inside a cloud
- Example StarTrek II - The Wrath of Khan
  - Genesis demo – first real particle system
  - Simulation of „Terraforming“
    - Display of a bomb impact on a planet
    - Propagation of an explosion wave
  - Hierarchical particle system
    - Top system – bomb impact
    - Sub system – fire walls





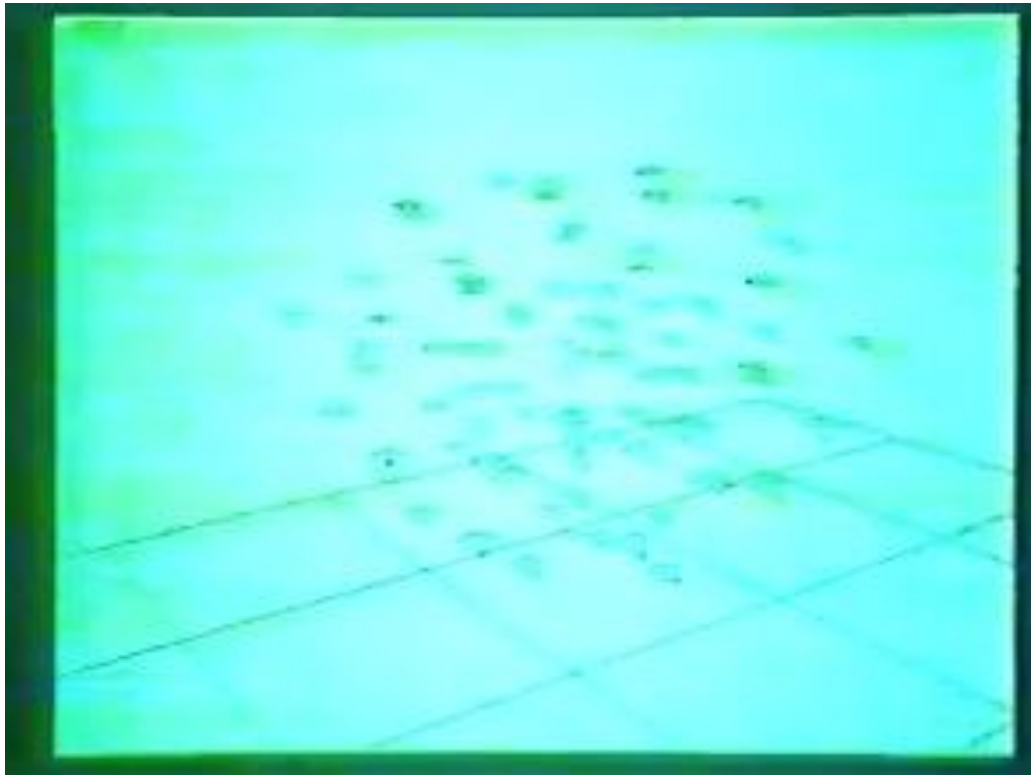
W. T. Reeves, “Particle Systems—a Technique for Modeling a Class of Fuzzy Objects,” *ACM Transactions on Graphics*, vol. 2, no. 2, pp. 91–108, Apr. 1983.

https://www.youtube.com/watch?v=52XlyMbxxh8

Particle Systems

- Implementation strategies
  - Model particles as structs instead of objects
  - When creating particles do not reserve memory and free it after particle destruction
  - Instead use reusable particle pools, the size of the pool should be the memory required for the maximum amount of particles
  - Can be implemented with single linked lists or double linked lists
  - Pool of unused particles should be available
  - Initialisation can be time consuming, initialise systems at application start
- Scene graphs and game engines
  - Different scene graphs and game engines support particle systems (OpenSG, OpenSceneGraph, Unity, Unreal)
  - Often complex parameterisation tools are used

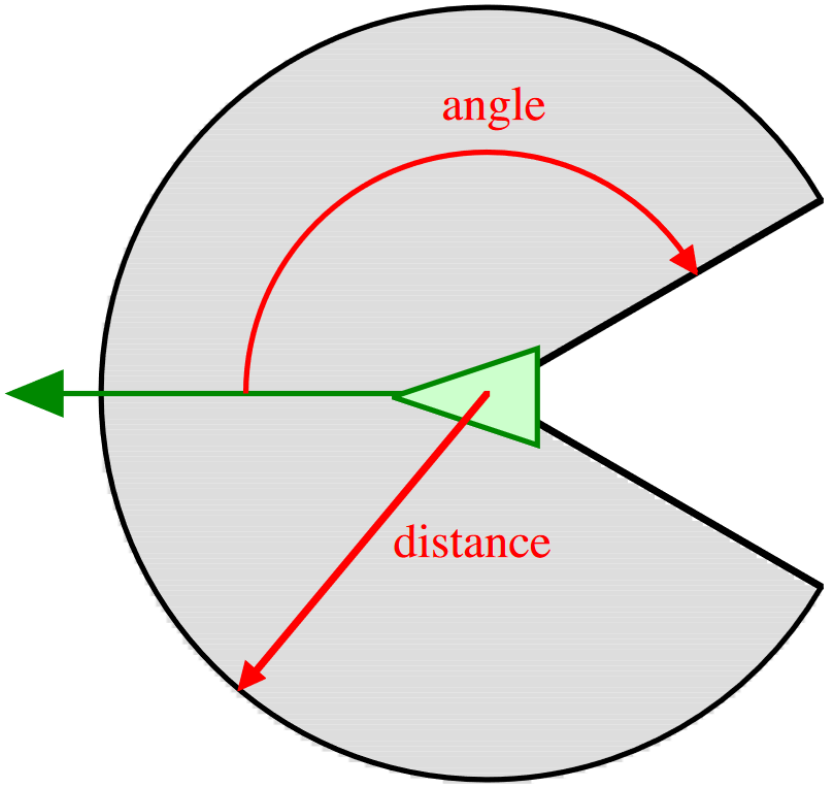
Swarming and Flocking



- “The motion of a flock of birds is... simple in concept yet is so visually complex it seems randomly arrayed and yet is magnificently synchronous. Perhaps most puzzling is the strong impression of intentional centralised control. Yet all evidence indicates that flock motion must be merely the aggregate result of the actions of individual animals, each acting solely on the basis of its local perception of the world.”
  - Craig Reynolds
- Craig Reynolds 1987 “Boids“ – “Bird-oid” objects
- Developed for simulation of flocks of birds, herds and swarms

Swarming and Flocking

- Attributes of a Boid
  - Boids know the attributes of their neighbours
    - Distance
    - Speed
    - Position
    - Orientation
  - Boids only consider their neighbourhood
- Neighbourhood of a Boid
  - Defined through angle and distance (visual range)
  - Different possibilities of neighbourhood definition possible
    - Elliptical shape
    - Neighbourhood increases if no neighbours are visible
    - Angle changes depending on simulated life form



Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Swarming and Flocking

- Differentiation from particle systems
  - Boids consider their neighbourhood
  - Boids do not have a defined life span
  - Behaviour of boids is independent from time
  - Boids are typically represented by complex geometries having their own coordinate system
  - Boids behave more complex than ordinary particles
  - No system core – no central control unit

- Force keeping the swarm together
  - Speed has to be adapted to the neighbour speed to keep the swarm together
  - Repulsion of individuals swarm members to avoid collisions
  - Decentralised motion, no leading element only consideration of the closer environment

Swarming and Flocking

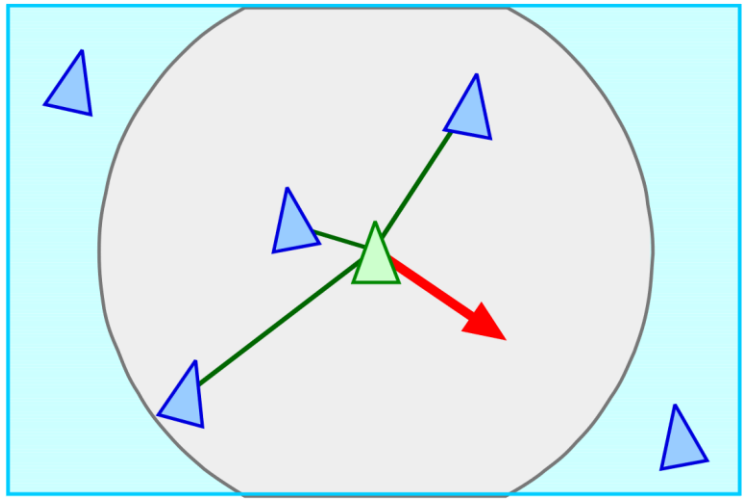
- Potential issues
  - Boids could collide
  - Boids could stop their movement
- Solutions
  - Different weighting of the calculated vectors
  - Expert systems for collision avoidance

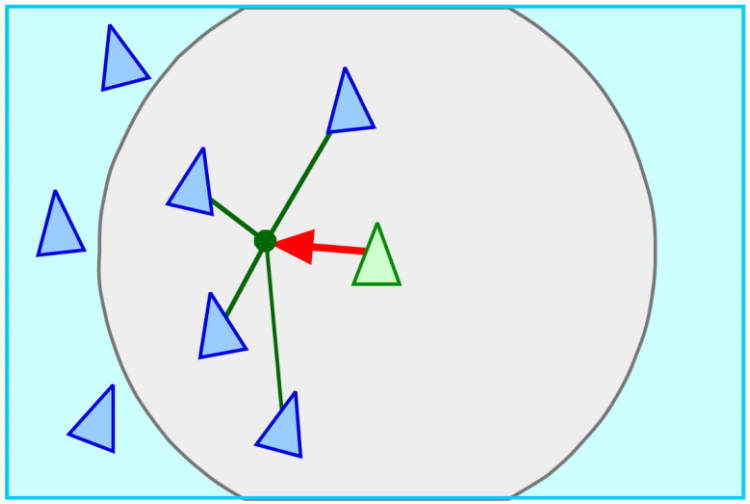
- Calculation of the direction vector is relatively complex
- Various factors influence the direction of movement of the Boid which result in direction vectors
- The new direction vector is calculated from these vectors

https://www.youtube.com/watch?v=eXZLmh6KJPQ

Swarming and Flocking

- Separation
  - Collision with neighbouring swarm members should be avoided
  - Position of neighbours is considered - not their speed
  - For each neighbour a repulsive force vector is calculated
  - The repulsive forces are summed up to indicate the a steering vector
- Cohesion
  - Trying to get to the centre of gravity of the neighbours
  - Again only position of neighbours is considered
  - Steering vector is calculated toward the centre of gravity
  - Cohesion of the swarm is thereby ensured

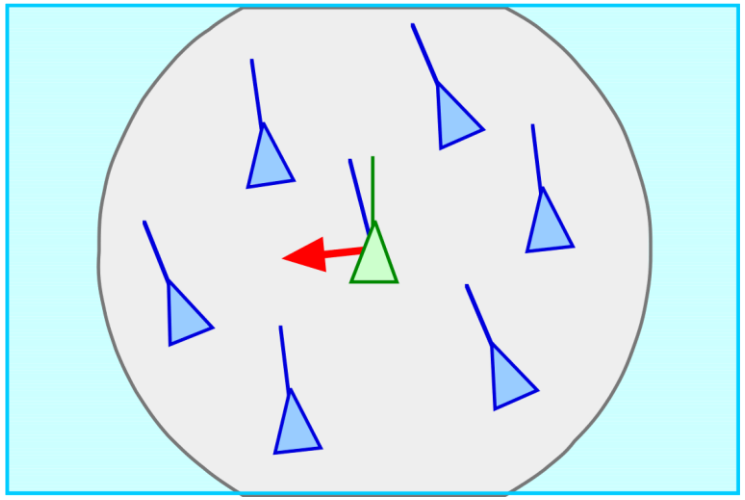


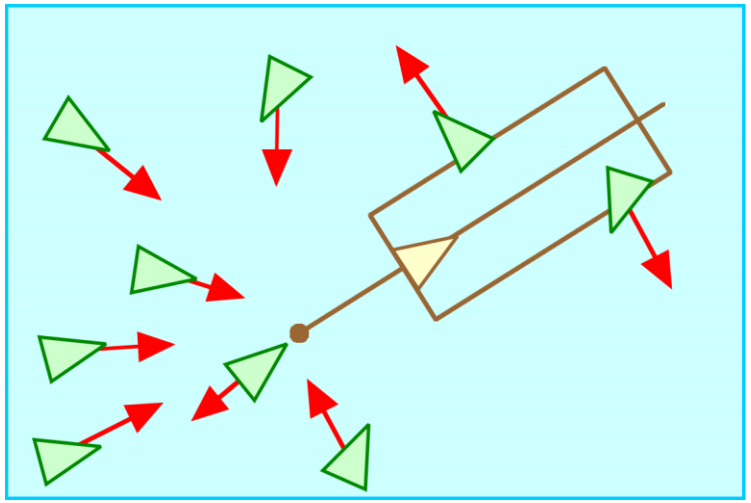


Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Swarming and Flocking

- Alignment
  - Try to adapt speed to neighbours
  - Averaging together the velocity (or alternately, the unit forward vector) of the neighbours resulting in a desired velocity
  - Steering vector is the difference between the averaged velocity vectors and the current velocity vector of the boid
- Follow the lead (extension of classical flocking)
  - Desired velocity vector is calculated pointing towards a point offset slightly behind the leader clamped by a maximum velocity
  - Steering vector is the difference between desired and actual velocity vector
  - If boids are to close or in front of target separation is used





Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Swarming and Flocking

- Simulation of swarms of bees
  - In the velocity matching (alignment) calculation, only the speed of the neighbours is taken into account, not the orientation
  - Behaviour of the boids is therefore much more chaotic

- Implementation
  - Boids should be modelled as an objects
  - Behaviours can be stored in member functions
  - For better control the three steering components should be normalised and the scaled according three weighting factors

Steering Behaviours

- Additional steering can be incorporated into swarming and flocking models
- Locomotion
  - Controlled by steering layer
  - Steering layer and locomotion layer can be completely independent
  - Movement representation can be implemented in different ways
    - Inverse kinematics
    - Static geometry
    - Motion cycle
- Vehicle model
  - Simplified vehicle model is used
  - High level of abstraction (cars, people, animals)
  - Vehicle model having local coordinate system adapted to direction of motion
  - Movement through incremental change to last calculation step

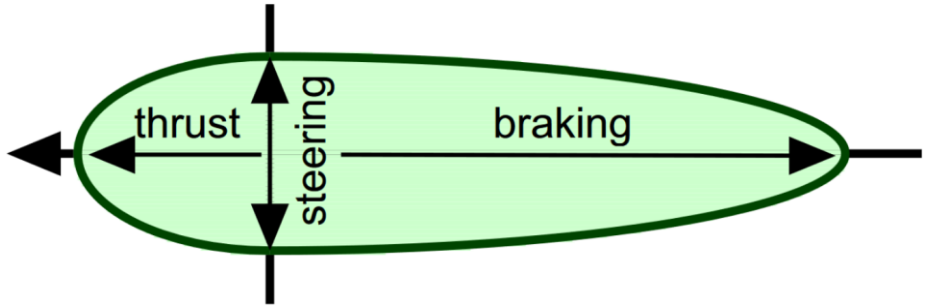
Steering Behaviours

- Vehicle model
  - Model serves as an approximation to any moving objects
  - A point mass is assumed
- Attributes
  - **`mass`** - (scalar)
  - **`position`** - (vector)
  - **`velocity`** - (vector)
  - **`max_force`** - maximum steering force (scalar)
  - **`max_speed`** - maximum speed (scalar)
  - **`orientation`** - (N base vectors)
- Undefined attributes such as position, velocity and orientation are defined once per simulation step

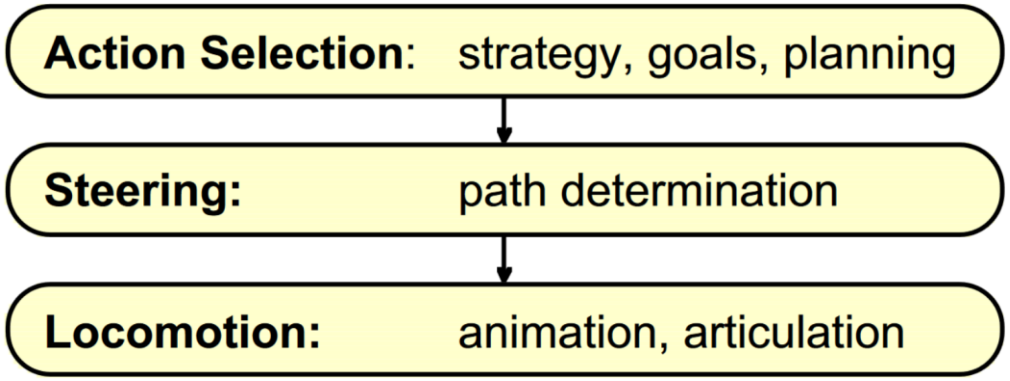
Steering Behaviours

- Calculation of the new position
  - Steering force is determined depending on steering behaviour (limited by maximum force)
  - Calculation of acceleration by steering force / mass
  - Speed results from speed + acceleration (limited by maximum speed)
  - Position results from position + speed

- Vectors can be scaled differently
- Asymmetrical steering forces can exist



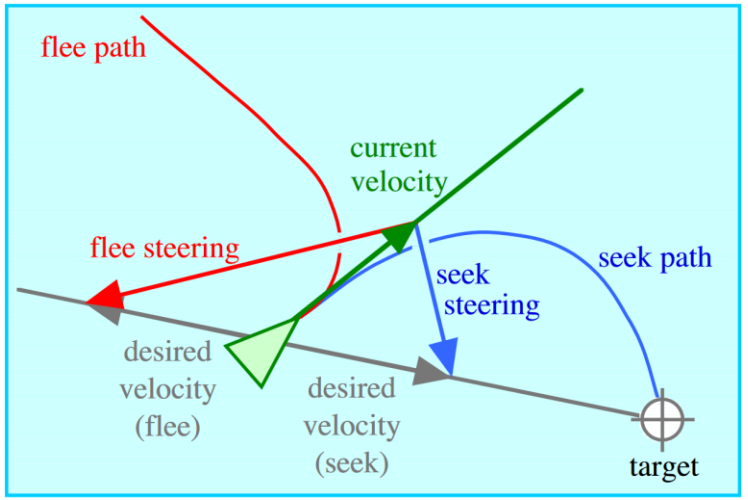
Behaviour of characters

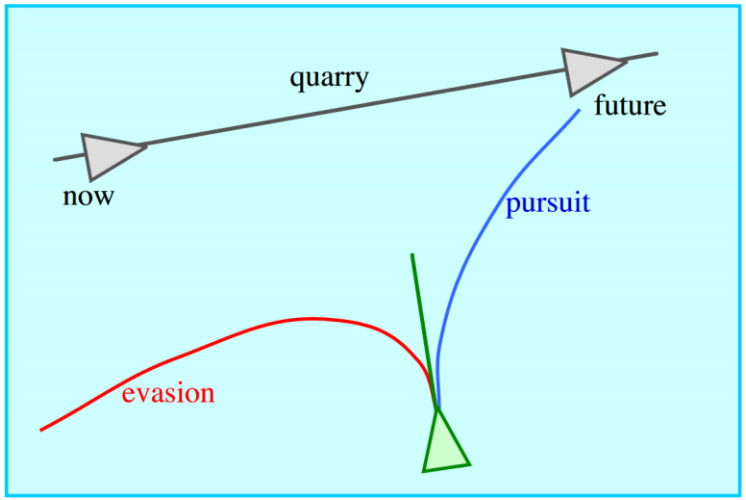


Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Steering Behaviours

- Seek and flee
  - Velocity is radially aligned towards the target
  - Steering vector is the difference between desired and actual velocity vector
  - Desired velocity could be max speed or user defined
  - Flee is calculated inverse to seek
- Pursuit and evasion
  - Similar to seek and flee behaviour but with moving target
  - Problem of predicting the future position of the target
    - Assumption the target does not change direction
    - Short distance to destination → short time interval
    - Long distance to target → use of a long time interval

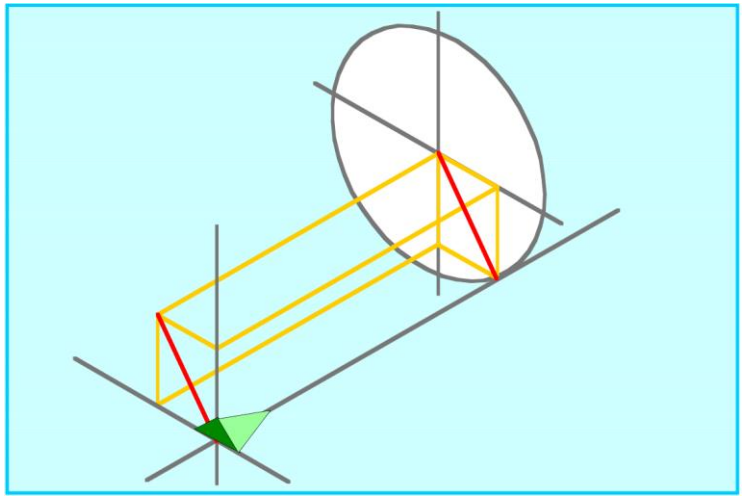


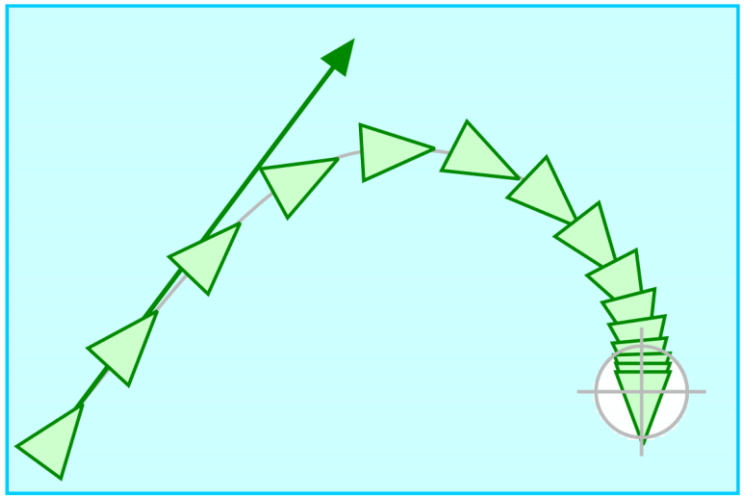


Craig W. Reynolds, "Steering Behaviors For Autonomous Characters", Game Developers Conference, pp 763-782,1999

Steering Behaviours

- Offset Pursuit
  - Used for fly-by or overtaking behaviour
  - Dynamically compute a target point which is offset by a given radius R from the predicted future position
  - Calculation with projection from vehicle coordinate system on target
  - Use of the target point in combination with seek behaviour
- Arrival
  - Identical to seek while the vehicle is far from its target
  - Stopping radius defined inside arrival behaviour
  - If target is outside speed is clamped to max_speed, otherwise speed is slowed down if vehicle come closer to target
  - Can lead to stopping if target point is reached

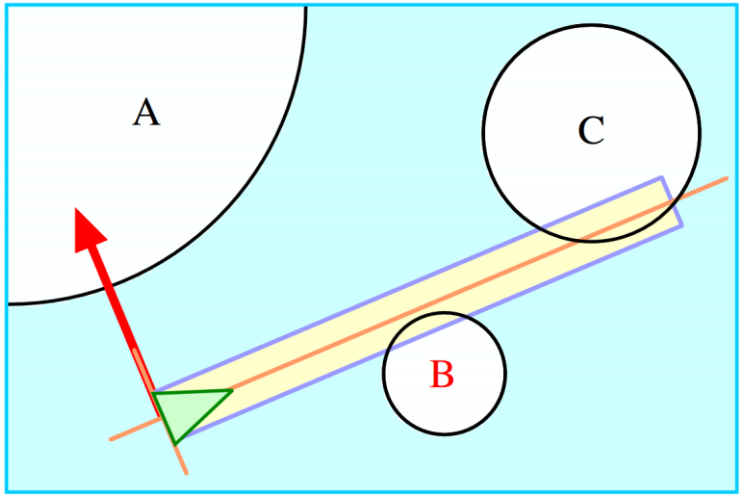


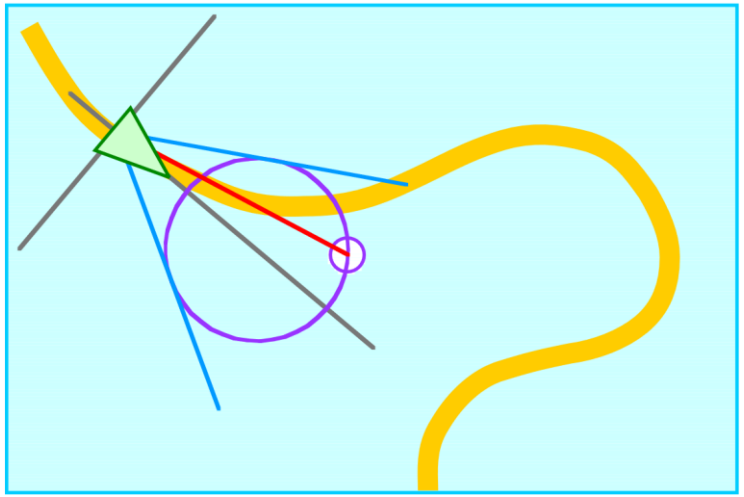


Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Steering Behaviours

- Obstacle avoidance
  - Manoeuvring around objects in environment
  - Vehicle only evades when object in way
  - The aim of the vehicle is to keep a empty cylinder in its path
  - Speed is used to predict position
- Wander
  - Random type of control
  - Random values too chaotic, more orderly procedure desired
  - Implementation idea
    - Ball in front of vehicle
    - Random values move target to sphere surface
    - Spherical radius limits maximum strength of travel
  - Variety of implementation strategies possible

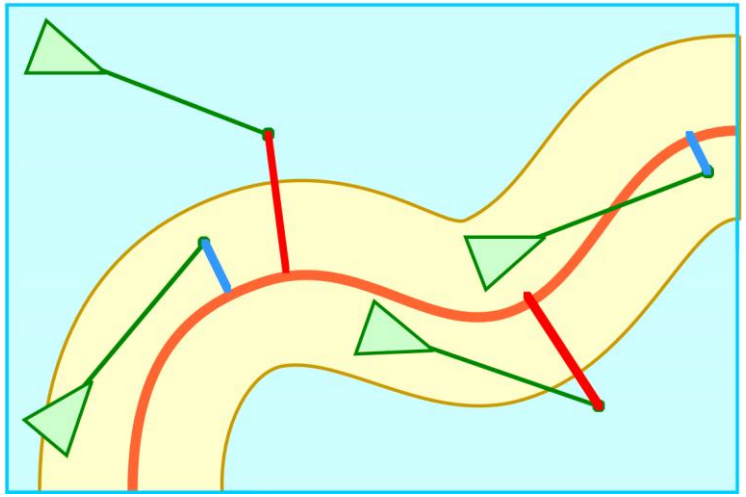


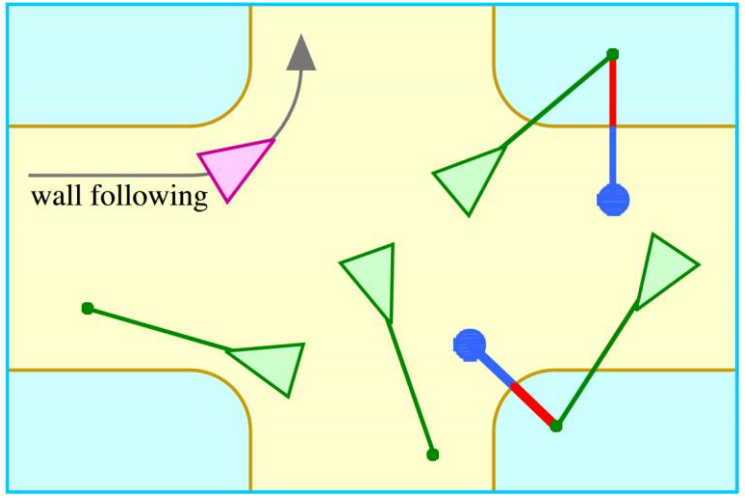


Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Steering Behaviours

- Path following
  - Vehicle follows a predefined path without direct connection to the path
  - Speed is used to predict position
  - Projection of the predicted point onto the path
  - If the distance is smaller than the radius of the path no control correction otherwise use seek steering behaviour
- Wall following / Containment
  - Vehicle must remain within certain range from walls
  - Speed is used to predict position
  - Same procedure as for path following

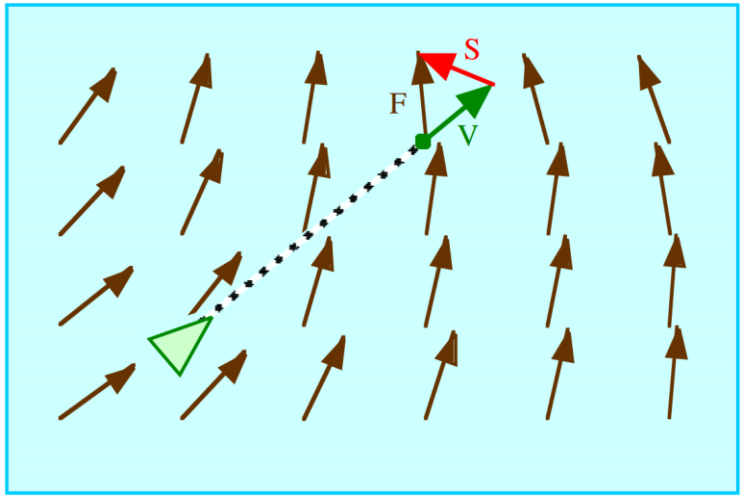


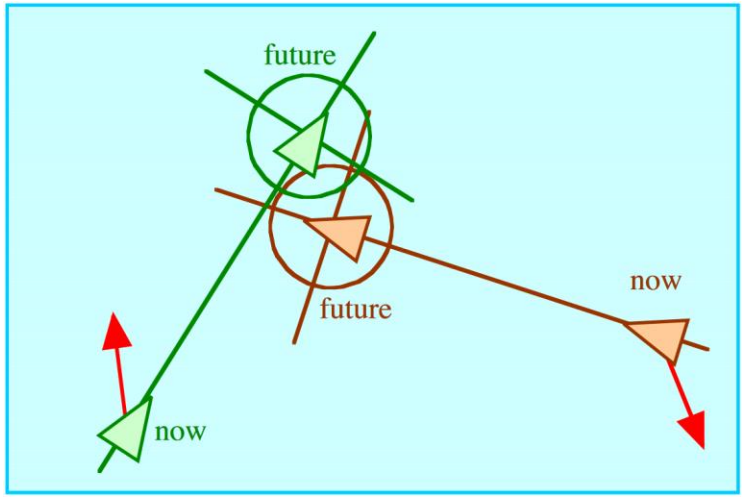


Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Steering Behaviours

- Flow following
  - Use of vector field
  - Speed is used to predict position
  - Force field vector at predicted position is determined
  - Force field vector represents the desired velocity
  - The steering vector results from the difference between the current velocity vector and the force field vector
- Unaligned collision avoidance
  - Prediction of a potential collision
  - Avoidance by steering behaviour to avoid the possible collision point
  - Steering vector opposite the predicted collision point





Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Bibliography

- N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann, “Joint-dependent local deformations for hand animation and object grasping,” *Proceedings of Graphics Interface ’88, Edmonton, Alberta, Canada, 6 - 10 June 1988, 26-33*, 1988.
- W. T. Reeves, “Particle Systems—a Technique for Modeling a Class of Fuzzy Objects,” *ACM Transactions on Graphics*, vol. 2, no. 2, pp. 91–108, Apr. 1983.
- Karl Sims, “Particle Animation and Rendering Using Data Parallel Computation“, ACM Transactions on Computer Graphics, 1990, 24, pages 405-413
- C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, Aug. 1987.
- Sederberg, T. W. & Parry, S. R. Free-form deformation of solid geometric models *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘86, ACM Press,* 1986
- Robin Green, “Steering Behaviors“, Course at SIGGRAPH 2000
- Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Bibliography

- Galvane, Q.; Christie, M.; Ronfard, R.; Lim, C.-K. & Cani, M.-P. Steering Behaviors for Autonomous Cameras Proceedings of Motion on Games - MIG'13, ACM Press, 2013

- Movement of autonomous characters in graphical environments
  - Craig Reynolds, „Steering Behaviors For Autonomous Characters“
    - http://www.red3d.com/cwr/papers/1999/gdc99steer.html
  - Clint Hannaford, Library for Autonomous Character Locomotion
    - http://www.atomicmedia.com/autonomous/
  - OpenSteer
    - http://opensteer.sourceforge.net/
  - Robin Green, Dungeon Keeper 2 SIGGRAPH 2000
    - http://www.red3d.com/siggraph/2000/course39/

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Computer Graphics

Outlook and Bibliography

FH-Prof. Dr. techn. Dipl. Inf. (FH) Christoph Anthes, MSc

Professor of Augmented and Virtual Reality

Overview

- What we have learned …
  - In the lecture
  - In the labs
- What we did not cover …
- Retrospective
  - Considering Labs
- Exam
- Theses
- Bibliography

What we have learned … in the lecture

- Introduction
  - General Concepts
  - History
- Anatomy of the Eye
  - Colour Perception
- Colour Models
  - CIE
  - RGB/RGBA
  - CYMK
  - HSL/HSV
  - Conversion Between Colour Models
- Geometric Primitives
  - Meshes
  - Triangles
  - Examples
- Transformations
  - Multiplying Vectors and Matrices
  - Transformations in 2D Space
  - Concatenating Transformations
  - Transformations in 3D Space
  - Quaternions and Normal Vectors
- Scene Graphs

What we have learned … in the lecture

- Projections
  - Frustum
  - Perspective Projection
  - Parallel Projection
- Clipping
  - Cohen-Sutherland Algorithm
- Culling Techniques
  - Frustum Culling
  - Backface Culling
  - Occlusion Culling
- Light Sources
  - Types of Light Sources
- Illumination or Reflectance Models
  - Lambert/Diffuse, Phong
- Advanced Reflectance Models
  - BRDF
  - Subsurface Scattering
- Shading Models
  - Flat, Gouraud, Phong
- Transparencies

What we have learned … in the lecture

- Textures
  - Texture coordinates and mapping
- Texture Maps
  - Bump Mapping, Normal Mapping, Displacement Mapping
  - Environment or Reflection Mapping
- Generating Textures
  - Perlin Noise
- Rendering Textures
  - MIP Mapping
- Buffers
  - Z-Buffer or Depth Buffer
  - Shadow Mapping
- Rasterisation
  - Naïve Line Drawing
  - Bresenham
- Anti-Aliasing
  - Gupta-Sproull
  - Wu
- Global Illumination
  - Raytracing
  - Radiosity
  - Volume Rendering

What we have learned … in the lecture

- Animation
- Skeletal Animation
  - Free Form Deformation
- Particle Systems
- Swarming and Flocking
- Steering Behaviours
  - Vehicle Model
  - Behaviours

What we have learned … in the labs

- Setup and Installation of Visual Studio
- OpenGL and NuGet
- Hello World!
- Basic GLUT Setup
- Input from Keyboard and Mouse
- Screen Space Operations with Scissors (optional)
- Drawing Meshes
- Projections
  - Perspective Projection
  - Orthographic Projection (optional)
- Geometrical Templates
- Transformations
  - The Matrix Stack
- Moving the Camera
- Animations

What we have learned … in the labs

- Basic Lighting
  - Colour and Ambient Light
- Lighting
  - Directional Light
  - Point Light
  - Spot Light
- Shading Models
- Materials
- Calculation of Colours in OpenGL (optional)
- Fog

- Creating Textures
- Initialising Textures
- Mapping Simple Textures
- Loading Textures

What we did not cover

- **Actually a fair bit**
- Display hardware
- Shader programming
- Deferred shading
- Advanced CG math
- Tricks with matrices
- Detailed camera models
- Perceptual concepts
- Graphics design
- Splines and NURBS
- Generative terrain, water simulation
- Fractals
- Stereoscopic rendering
- Graphical effects (e.g. bloom, lens flares, …)
- Constructive geometry
- Non-Photorealistic Rendering
- Ambient Occlusion
- Physics
- Fluids

Retrospective

- Roughly 90% of the concepts introduced in this course are implemented in modern game and rendering engines or at least partially in 3d modelling tools like Unity, Unreal, 3DS Max or Cinema 4d
- Most game engines at least provide roughly 90% more CG in terms of algorithms, features and concepts
- You have gained a **rough overview** of CG and will most likely know much more than an ordinary Unity or Unreal developer about the inner workings of these systems, but still **do not** consider yourself as a CG expert
- You might consider parts of the lecture ‘retro’ since it was a fundamental lecture, but feel free to ask for more lectures in this area which will look into shaders, Programmable Pipeline OpenGL, high-end rendering, etc.
- More playful and high-level courses are the AR and VR courses ;)

Retrospective – Considering Labs

- We explored a bit of the OpenGL API
  - You should be able to dig a bit deeper into OpenGL
  - You should be able to develop a rudimentary rendering engine (if you put a lot of time in)
  - You should be able to code with graphics APIs in a reasonable way
- As with the lecture you have gained **a rough overview** of OpenGL, **do not** consider yourself as an OpenGL expert

- When you want to pursue working with OpenGL and you want to get into the details switch to modern OpenGL (4.5 and above)
- If you do not want to go that deep into OpenGL but need more flexibility try working with GLSL (Graphics Library Shading Language)

Exam

- Most of the questions will not be reproduction of the slides, they will require you to reproduce the knowledge gained in the course

- Examples:
  - When two planes are co-aligned in 3d space, what might happen? Reason why?
  - If you see this triplet (128, 20, 90) in the context of a colour definition, what could it mean? What colour could it describe?
  - Take a look at the matrices, in what order are they multiplied to generate the resulting image
  - What will happen if you apply the following matrix on the line leading from (1,0) to (2,0)
  - Comparing projections if you have a parallel projection what would be the difference to a perspective projection, when you render a cube?

Exam

- Examples
  - Assuming you would want to render flat shading but you are in a Gouraud shading mode. You are free to change all attributes of the triangle representation. What would you have to do?
  - What is the difference between normal mapping and displacement mapping in terms of rendering quality?
  - Take a look at this rendering is this global or local illumination? Reason why.
  - Take a look at this equation. What is it used for in terms of reflectance?
  - Describe the structure of a particle system
  - What is the difference between particle systems and swarming and flocking algorithms?
  - Considering skeletal animation, in what way would it be relevant for particle systems or steering algorithms?

Exam

- Exam might be an on site Moodle Test with parts multiple choice and parts free text. Alternatively a classical written exam.

- What is **not** going to be part of the exam
  - OpenGL
  - History of graphics
  - Advanced topics (Rendering Equation, BRDFs, Subsurface Scattering)
  - Simple reproduction of equations

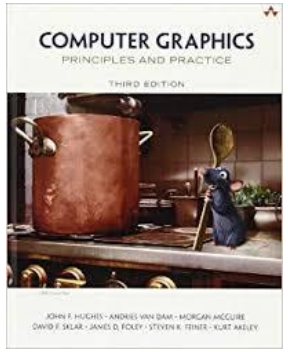
Theses

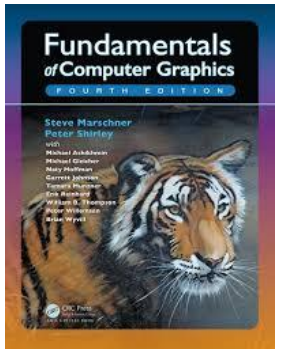
- If you have an idea considering a project for a Bachelor Thesis in the graphics, AR or VR domain feel free to ask
- Same is true when you think about Master Thesis topics later on
- The topic has to have a research question
- There will be some topics in the catalogue, but I’m happy to discuss further topics with you

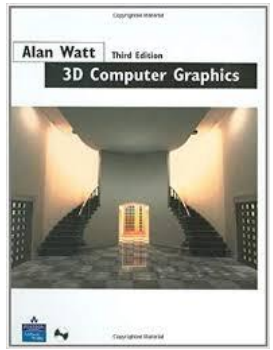
- If you are interested in internships in the graphics, AR, VR domain feel free to ask
- The internship connections I offer are partially industry related but also quite often in the research domain

Literature - Lecture

- J. F. Hughes et al., Computer Graphics: Principles and Practice (3rd Edition). Addison-Wesley Professional, 2013.
- S. Marschner and P. Shirley, *Fundamentals of Computer Graphics, Fourth Edition*. A K Peters/CRC Press, 2015.
- A. Watt, *3D Computer Graphics, 3rd Edition*. Addison-Wesley 1999.
- H.-J. Bungartz, M. Griebel, and C. Zenger, *Einführung in die Computergraphik: Grundlagen, Geometrische Modellierung, Algorithmen (Mathematische Grundlagen der Informatik) (German Edition)*. Vieweg+Teubner Verlag, 2002
- Zeppenfeld, K. Lehrbuch der Grafikprogrammierung: Grundlagen, Programmierung, Anwendung (German Edition) Spektrum Akademischer Verlag, 2003

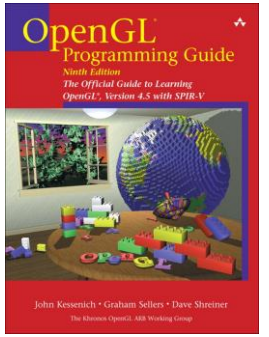




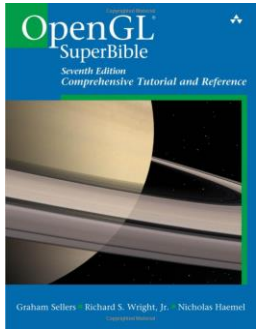


Literature - Labs

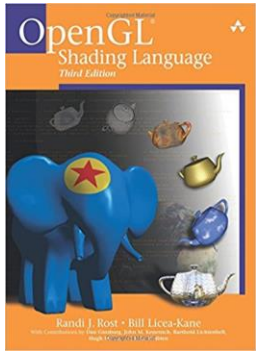
- Shreiner, D.; Sellers, G.; Kessenich, J. & Licea-Kane, B. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3 (8th Edition) *Addison-Wesley Professional,* 2013



- Sellers, G.; Jr., R. S. W. & Haemel, N. OpenGL Superbible: Comprehensive Tutorial and Reference (7th Edition) Addison-Wesley Professional, 2015



- Rost, R. J.; Licea-Kane, B.; Ginsburg, D.; Kessenich, J.; Lichtenbelt, B.; Malan, H. & Weiblen, M. OpenGL Shading Language (3rd Edition) Addison-Wesley Professional, 2009



Bibliography - Introduction, Colour Perception and Colour Models

- M. H. Weik, *Communications Standard Dictionary*. Van Nostrand Reinhold Company, 1982.
- William Fetter, *Computer Graphics at Boeing.* In: *Print Magazine*, XX:VI, November/Dezember 1966, p. 32
- A. van Dam CSC1 1230 Introduction into Computer Graphics, Brown University 2018
- M. Levoy, History of computer graphics, slideset CS 248 - Introduction to Computer Graphics Autumn quarter, 2004, Slides for September 28 lecture, Stanford
- I. E. Sutherland, “Sketchpad: A man-machine graphical communications system,” MIT Lincoln Laboratories, Technical Report 296, 1963.
- L. G. Roberts, “Machine Perception Of Three-Dimensional Solids,” Lincoln Laboratory, TR 315, MIT, Cambridge, 1963.
- A. Appel, “The notion of quantitative invisibility and the machine rendering of solids,” in *Proceedings of the 1967 22nd national conference on* -, 1967.
- J. Warnock, “A Hidden-Surface Algorithm for Computer Generated Halftone Pictures,” Computer Graphics Department, University of Utah, Salt Lake City, Technical Report TR 4-15, NTIS AD-753 671, 1969.

Bibliography - Introduction, Colour Perception and Colour Models

- G. S. Watkins, “A real time visible surface algorithm,” The University of Utah, 1970.
- E. E. Sutherland, R. F. Sproull, and R. A. Schumacker, “A Characterization of Ten Hidden-Surface Algorithms,” *ACM Computing Surveys*, vol. 6, no. 1, pp. 1–55, Jan. 1974.
- H. Gouraud, “Continuous Shading of Curved Surfaces,” *IEEE Transactions on Computers*, vol. C-20, no. 6, pp. 623–629, Jun. 1971.
- J. F. Blinn, “Simulation of wrinkled surfaces,” *ACM SIGGRAPH Computer Graphics*, vol. 12, no. 3, pp. 286–292, Aug. 1978.
- E. E. Catmull, “A subdivision algorithm for computer display of curved surfaces.,” The University of Utah, 1974.
- F. C. Crow, “The aliasing problem in computer-generated shaded images,” *Communications of the ACM*, vol. 20, no. 11, pp. 799–805, Nov. 1977.
- Whitted, T. An improved illumination model for shaded display Communications of the ACM, Association for Computing Machinery (ACM), 1980, 23, 343-349
- Cook, R. L. Shade trees Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘84, ACM Press, 1984

Bibliography - Introduction, Colour Perception and Colour Models

- Kajiya, J. T. The rendering equation ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1986, 20, 143-150
- W. T. Reeves, “Particle Systems—a Technique for Modeling a Class of Fuzzy Objects,” *ACM Transactions on Graphics*, vol. 2, no. 2, pp. 91–108, Apr. 1983.
- C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, Aug. 1987.
- N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann, “Joint-dependent local deformations for hand animation and object grasping,” *Proceedings of Graphics Interface ’88, Edmonton, Alberta, Canada, 6 - 10 June 1988, 26-33*, 1988.
- Haeberli, P. Paint by numbers: abstract image representations Proceedings of the 17th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘90, ACM Press, 1990
- Drebin, R. A.; Carpenter, L. & Hanrahan, P. Volume rendering Proceedings of the 15th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘88, ACM Press, 1988

Bibliography - Introduction, Colour Perception and Colour Models

- Jones, A.; McDowall, I.; Yamada, H.; Bolas, M. & Debevec, P. Rendering for an interactive 360degree light field display *ACM Transactions on Graphics, Association for Computing Machinery (ACM),* 2007*, 26*, 40
- Winkenbach, G. & Salesin, D. H. Computer-generated pen-and-ink illustration Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘94, ACM Press, 1994
- Meier, B. J. Painterly rendering for animation Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘96, ACM Press, 1996
- D. Purves *et al.*, Eds., *Neuroscience (Book with CD-ROM)*. Sinauer Associates Inc, 2001.
- A. V. Oppenheim, R. W. Schafer, and T. G. Stockham, “Nonlinear filtering of multiplied and convolved signals,” *Proceedings of the IEEE*, vol. 56, no. 8, pp. 1264–1291, 1968.

Bibliography - Modelling and Transformations

- James F. Blinn, “Jim Blinn’s Corner: Nested transformations and blobby man“, *IEEE Computer Graphics and Applications,* 1987*, 7*, pages 65-69
- Avi Bar-Zeev, “Scenegraphs: Past, Present and Future“, last visited February ‘19 http://www.realityprime.com/articles/scenegraphs-past-present-and-future

- Scene Graphs - Example applications
  - Open Inventor (https://www.openinventor.com/)
  - OpenSceneGraph (http://www.openscenegraph.org/)
  - OpenSG (https://sourceforge.net/projects/opensg/)
  - Unity (https://unity3d.com)
  - Unreal (https://www.unrealengine.com/)

Bibliography - Projections, Clipping, and Culling

- Carlbom, I. & Paciorek, J. Planar Geometric Projections and Viewing Transformations ACM Computing Surveys, Association for Computing Machinery (ACM), 1978, 10, 465-50
- James, O.; von Tunzelmann, E.; Franklin, P. & Thorne, K. S. Gravitational lensing by spinning black holes in astrophysics, and in the movie Interstellar Classical and Quantum Gravity, IOP Publishing, 2015, 32, 065001
- Sutherland, I. E. & Hodgman, G. W. Reentrant polygon clipping Communications of the ACM, Association for Computing Machinery (ACM), 1974, 17, 32-42
- Cohen-Or, D.; Chrysanthou, Y.; Silva, C. & Durand, F. A survey of visibility for walkthrough applications IEEE Transactions on Visualization and Computer Graphics, Institute of Electrical and Electronics Engineers (IEEE), 2003, 9, 412-431
- Papaioannou, G.; Gaitatzes, A. & Christopoulos, D. Efficient Occlusion Culling using Solid Occluders WSCG'2006, 2006
- Fuchs, H.; Kedem, Z. M. & Naylor, B. F. On visible surface generation by a priori tree structures ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1980, 14, 124-133

Bibliography - Illumination, Reflectance and Shading Models

- Kajiya, J. T. The rendering equation ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1986, 20, 143-150
- Johann Heinrich Lambert, Ernst Anding: Lamberts Photometrie: 1. Th. Das directe Licht. - 2. Th. Die Schwächung des Lichts durch durchsichtige Körper, besonders durch Glas. W. Engelmann, 1892
- Phong, B. T. Illumination of Computer-Generated Images, Department of Computer Science, University of Utah, UTEC-CSs-73-129, July 1973.
- Blinn, J. F. Models of light reflection for computer synthesized pictures ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1977, 11, 192-198
- Oren, M. & Nayar, S. K. Generalization of Lambert's reflectance model Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94, ACM Press, 1994
- Torrance, K. E. & Sparrow, E. M. Theory for Off-Specular Reflection From Roughened Surfaces Journal of the Optical Society of America, The Optical Society, 1967, 57, 1105

Bibliography - Illumination, Reflectance and Shading Models

- Nicodemus, F. E. Directional Reflectance and Emissivity of an Opaque Surface Applied Optics, The Optical Society, 1965, 4, 767
- Cohen, M. F. Radiosity and Realistic Image Synthesis Morgan Kaufmann, 2016
- Hanrahan, P. & Krueger, W. Reflection from layered surfaces due to subsurface scattering Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘93, ACM Press, 1993
- Debevec, P.; Hawkins, T.; Tchou, C.; Duiker, H.-P.; Sarokin, W. & Sagar, M. Acquiring the reflectance field of a human face Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘00, ACM Press, 2000
- Gouraud, H. Continuous Shading of Curved Surfaces IEEE Transactions on Computers, Institute of Electrical and Electronics Engineers (IEEE), 1971, C-20, 623-629
- Phong, B. T. Illumination for computer generated pictures Communications of the ACM, Association for Computing Machinery (ACM), 1975, 18, 311-317

Bibliography - Texture Mapping and Buffers

- Akenine-Möller, T.; Haines, E. & Hoffman, N. Real-time Rendering Taylor & Francis Ltd., 2008
- Blinn, J. F. Simulation of wrinkled surfaces ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1978, 12, 286-292
- Kaneko, T.; Takahei, T.; Inami, M.; Kawakami, N.; Yanagida, Y.; Maeda, T. & Tachi, S. Detailed Shape Representation with Parallax Mapping ICAT 2001, 2001
- Cook, R. L. Shade trees Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘84, ACM Press, 1984
- Blinn, J. F. & Newell, M. E. Texture and reflection in computer generated images Communications of the ACM, Association for Computing Machinery (ACM), 1976, 19, 542-547
- Cohen, J.; Olano, M. & Manocha, D. Appearance-preserving simplification Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘98, ACM Press, 1998
- Perlin, K. An image synthesizer Proceedings of the 12th annual conference on Computer graphics and interactive techniques - SIGGRAPH '85, ACM Press, 1985

Bibliography - Texture Mapping and Buffers

- Lagae, A.; Lefebvre, S.; Cook, R.; DeRose, T.; Drettakis, G.; Ebert, D.; Lewis, J.; Perlin, K. & Zwicker, M. A Survey of Procedural Noise Functions Computer Graphics Forum, Wiley, 2010, 29, 2579-2600
- Odaker, T.; Wiedemann, M.; Anthes, C. & Kranzlmüller, D. Texture analysis and repacking for improved storage efficiency Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology - VRST '16, ACM Press, 2016
- Williams, L. Casting curved shadows on curved surfaces Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH '78, ACM Press, 1978
- Edwin Catmull: A Subdivision Algorithm for Computer Display of Curved Surfaces. Dissertation, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City 1974

Bibliography - Rasterisation and Global Illumination

- Bresenham, J. E. Algorithm for computer control of a digital plotter IBM Systems Journal, IBM, 1965, 4, 25-30
- Wu, X. An efficient antialiasing technique ACM SIGGRAPH Computer Graphics, Association for Computing Machinery (ACM), 1991, 25, 143-152
- Gupta, S. & Sproull, R. F. Filtering edges for gray-scale displays Proceedings of the 8th annual conference on Computer graphics and interactive techniques - SIGGRAPH '81, ACM Press, 1981
- Appel, A. Some techniques for shading machine renderings of solids Proceedings of the April 30-May 2, 1968, spring joint computer conference on - AFIPS '68 (Spring), ACM Press, 1968
- Whitted, T. An improved illumination model for shaded display Communications of the ACM, Association for Computing Machinery (ACM), 1980, 23, 343-349
- Glassner, A. S. An Introduction to Ray Tracing ACADEMIC PR INC, 1989
- Jensen, H. W. Realistic Image Synthesis Using Photon Mapping, 2nd Edition A K PETERS LTD (MA), 2001

Bibliography - Rasterisation and Global Illumination

- Immel, D. S.; Cohen, M. F. & Greenberg, D. P. A radiosity method for non-diffuse environments Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘86, ACM Press, 1986
- Lorensen, W. E. & Cline, H. E. Marching cubes: A high resolution 3D surface construction algorithm Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87, ACM Press, 1987
- Akio Doi, Akio Koide. "An Efficient Method of Triangulating Equi-Valued Surfaces by Using Tetrahedral Cells." IEICE Transactions of Information and Systems, Vol.E74-D No. 1, 1991

Bibliography - Animation

- N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann, “Joint-dependent local deformations for hand animation and object grasping,” *Proceedings of Graphics Interface ’88, Edmonton, Alberta, Canada, 6 - 10 June 1988, 26-33*, 1988.
- W. T. Reeves, “Particle Systems—a Technique for Modelling a Class of Fuzzy Objects,” *ACM Transactions on Graphics*, vol. 2, no. 2, pp. 91–108, Apr. 1983.
- Karl Sims, “Particle Animation and Rendering Using Data Parallel Computation“, ACM Transactions on Computer Graphics, 1990, 24, pages 405-413
- C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, Aug. 1987.
- Sederberg, T. W. & Parry, S. R. Free-form deformation of solid geometric models *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH ‘86, ACM Press,* 1986
- Robin Green, “Steering Behaviors“, Course at SIGGRAPH 2000
- Craig W. Reynolds, “Steering Behaviors For Autonomous Characters“, Game Developers Conference, pp 763-782,1999

Bibliography - Animation

- Galvane, Q.; Christie, M.; Ronfard, R.; Lim, C.-K. & Cani, M.-P. Steering Behaviors for Autonomous Cameras Proceedings of Motion on Games – MIG'13, ACM Press, 2013

- Movement of autonomous characters in graphical environments
  - Craig Reynolds, „Steering Behaviors For Autonomous Characters“
    - http://www.red3d.com/cwr/papers/1999/gdc99steer.html
  - Clint Hannaford, Library for Autonomous Character Locomotion
    - http://www.atomicmedia.com/autonomous/
  - OpenSteer
    - http://opensteer.sourceforge.net/
  - Robin Green, Dungeon Keeper 2 SIGGRAPH 2000
    - http://www.red3d.com/siggraph/2000/course39/

Bibliography – Some other useful resources

- Real-Time Rendering Resources
  - http://www.realtimerendering.com/

- OpenGL Sample code
  - https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html