3D Concepts & Representation

Dept. Ilmu Komputer

2021

Learning Objectives

- 3D Concepts
- 3D Object Representations
  - Point cloud
  - Polyhedron
  - Quadrics
  - Spline curves represntation
  - Blobby Object
  - Constructive Solid Geometry
  - Fractal & Particle System

3D Concept

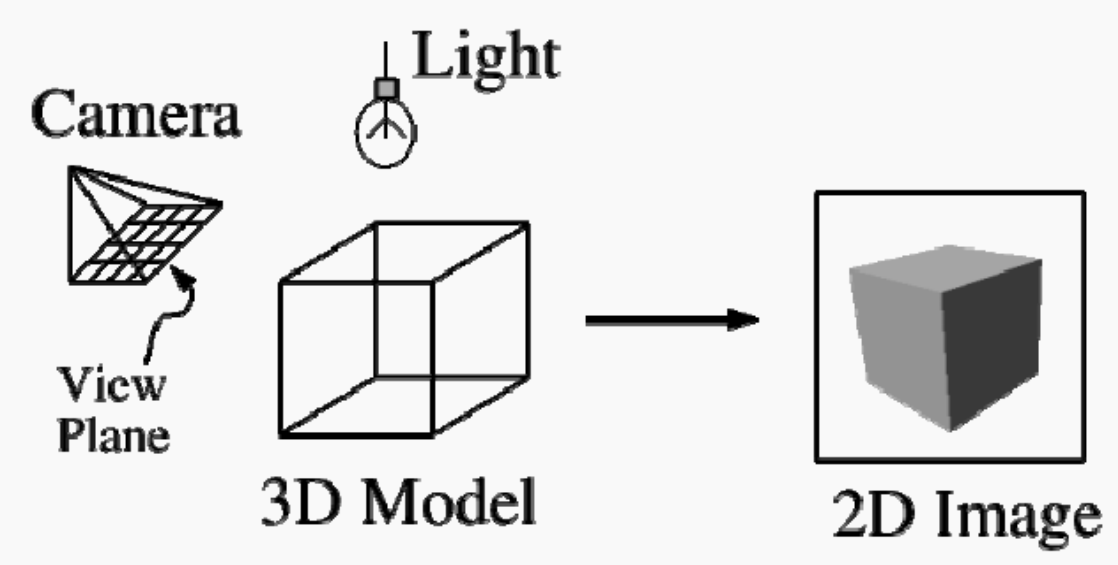
- 3D Object : Representasi dari data geometrik 3 dimensi sebagai hasil pemrosesan dari efek depth (cahaya, texture, dll) pada objek berdimensi 2 (flat object)
- Pembuatan objek gambar dengan menggunakan 3 titik sebagai acuannya yaitu sumbu x, y dan z yang kemudian ditinjau secara matematis dalam melihat suatu objek, dimana gambar tersebut dapat terlihat secara menyeluruh dan nyata.

3D Concept

- **Graphics scenes** melibatkan berbagai jenis objek dan material surfaces
  - Kursi. Meja, Lampu, lukisan, Buku, Monitor
- Bagaimana merepresentasikan objek 3D ?

3D Concept

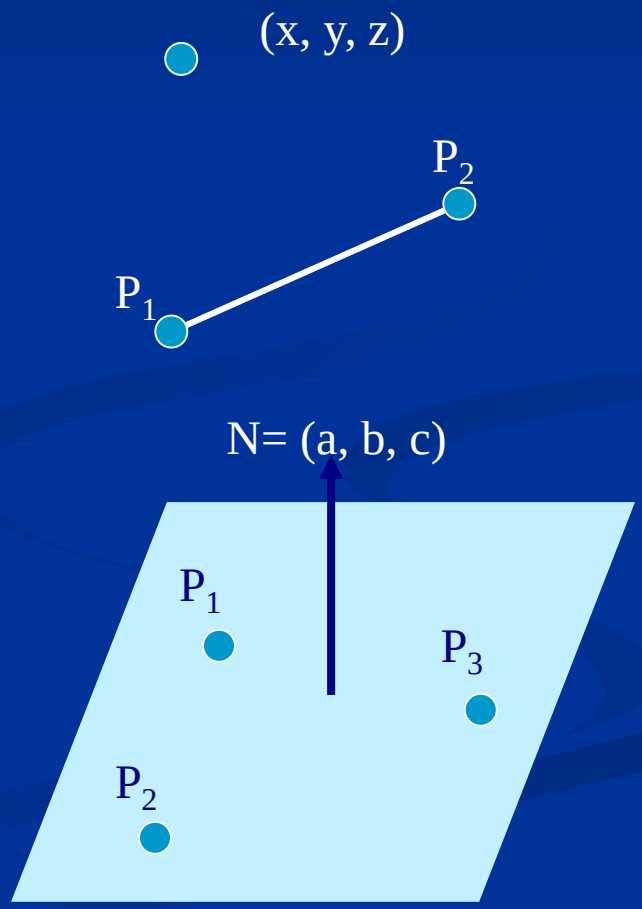
- Modeling : Merepresentasikan dimensi objek
- Rendering : Membentuk gambar 2D dari objek 3D
- Animation : Simulasi perubahan terhadap waktu



3D Concept

Primitif Geometri 3D

- Titik
  - Menandakan lokasi dalam dimensiruang
- Segmen garis
  - Kombinasi linear dua titik
- Bidang (plane)
  - Kombinasi linear tiga titik



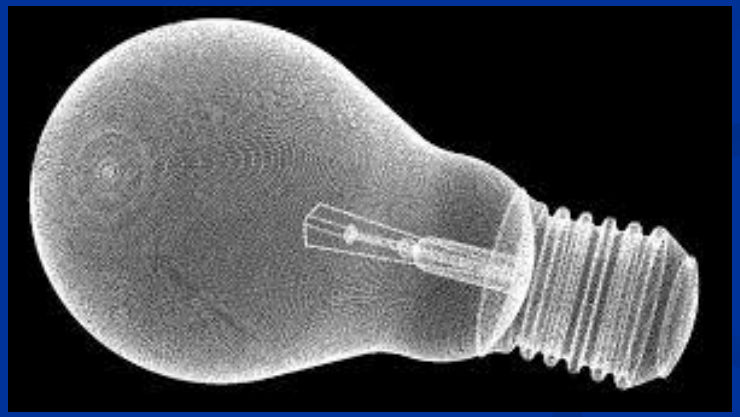
Teknik Representasi

- Dasar
  - Point cloud
  - Range image
- Permukaan
  - Polyhedron
  - Polygon mesh
  - Quadric
  - Bezier & spline
- Benda padat (solid)
  - Voxels
  - Sweep
  - CSG
- Advanced
  - Fraktal
  - Particle system
  - Physics-based

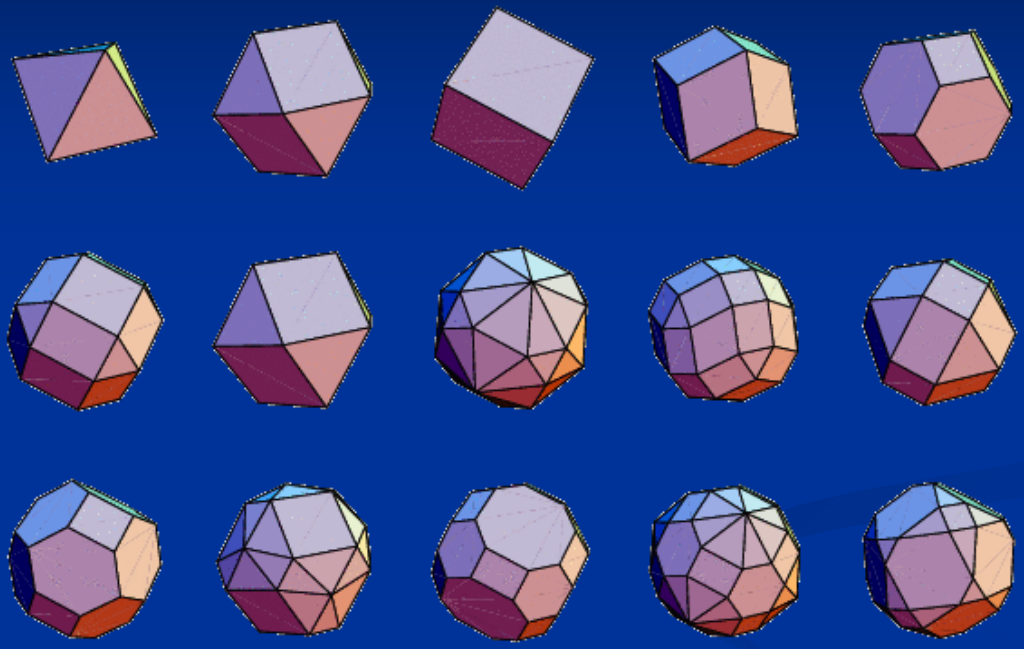
3D Object Representations

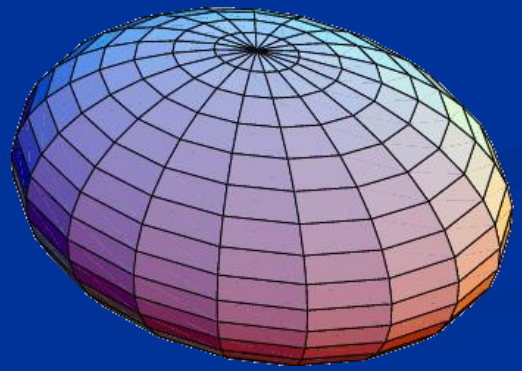
- Untuk merepresentasikan objek 3D, ada beberapa teknik
  - Membuat objek dalam representasi titik dalam ruang
  - Untuk membuat objek yang memiliki permukaan seperti polyhedrons ataupun ellipsoids maka menggunakan polygon dan quadric
  - Untuk membuat permukaan melengkung seperti pada sayap pesawat, gears, bodi mesin, etc, digunakan Spline surfaces
  - Untuk menyusun bentuk geometri dasar menjadi objek komplek dapat menggunakan teknik Constructive solid geometry
  - Untuk memodelkan pegunungan, awan, tumbuhan, atau air terjun digunakan *procedural methods* seperti fractals dan particle system

3D Object Representations

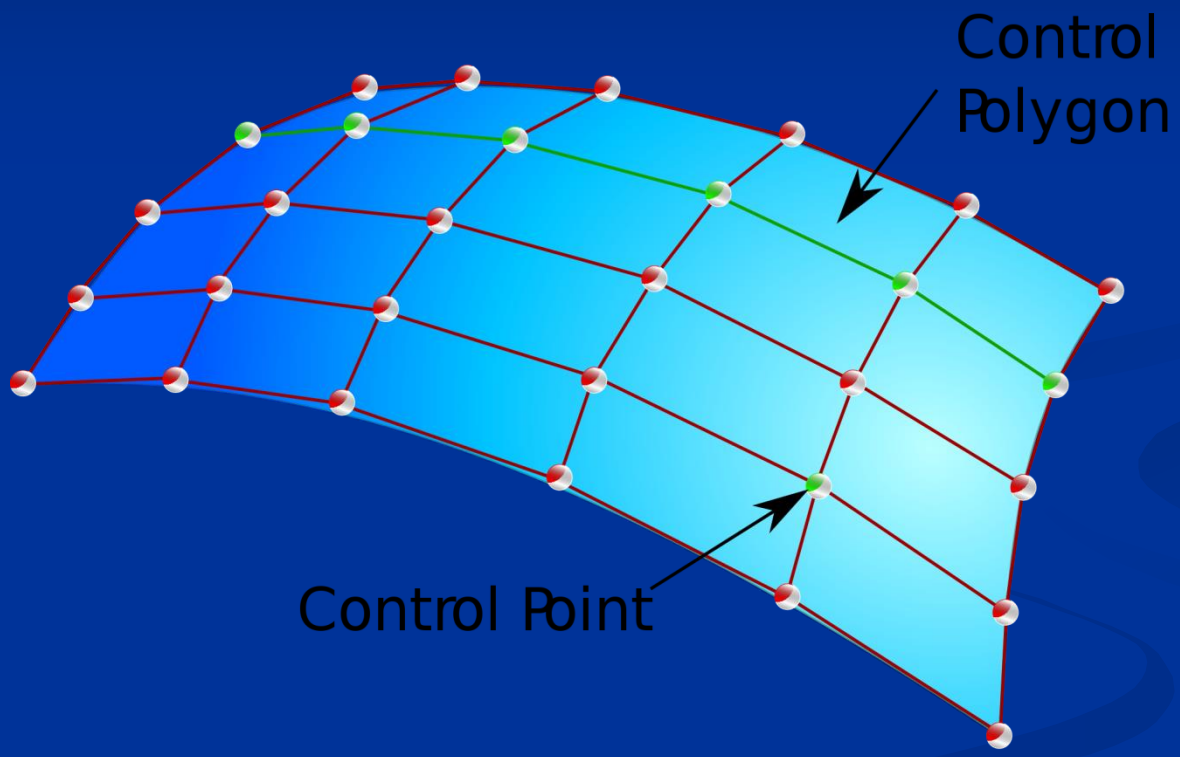


3D Object Representations

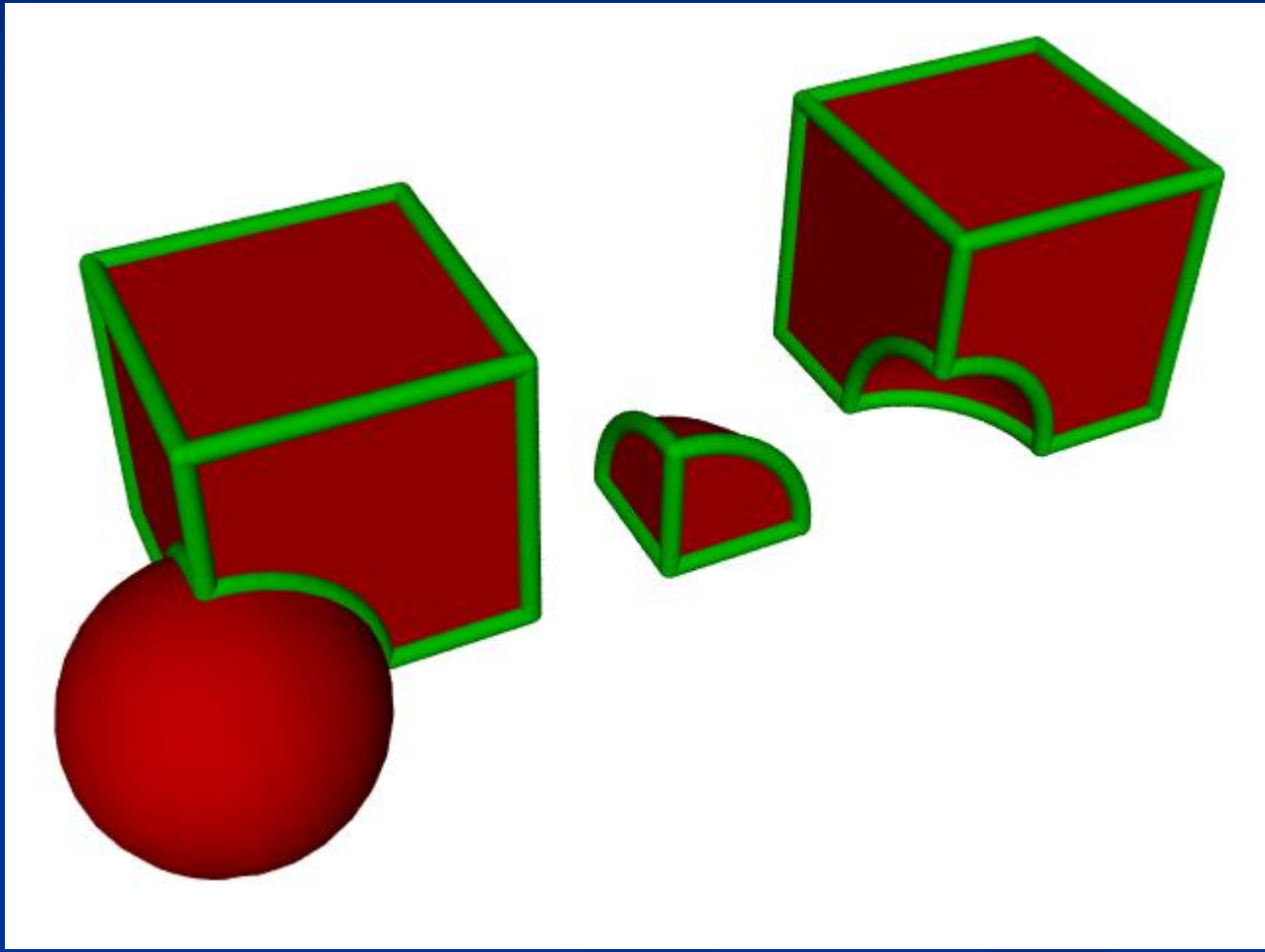




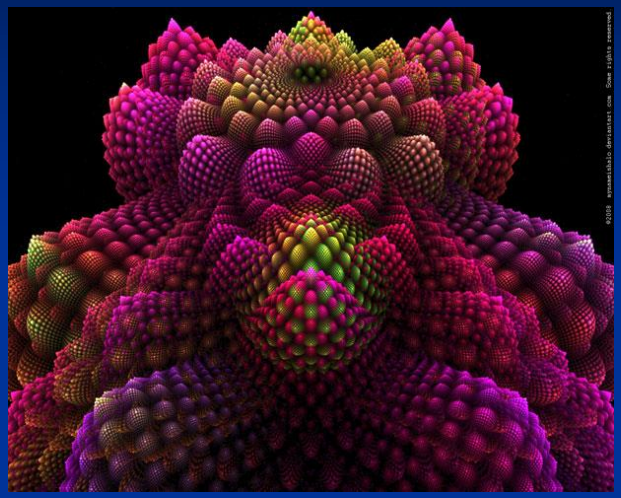
3D Object Representations

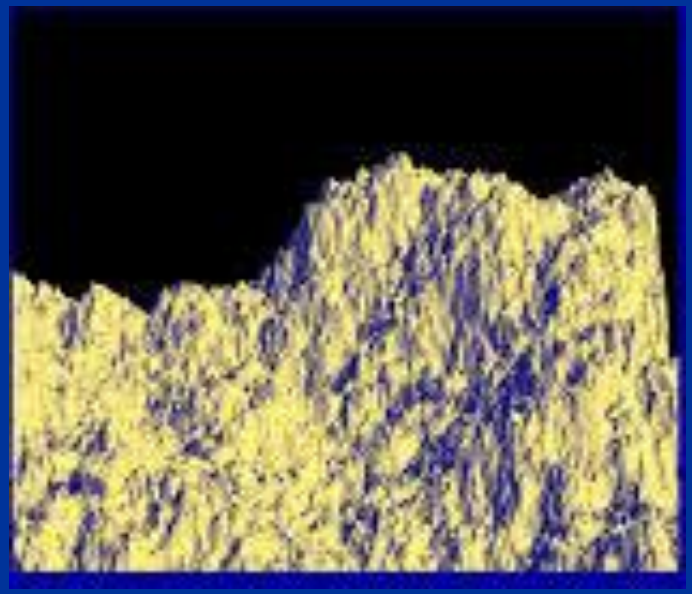


3D Object Representations



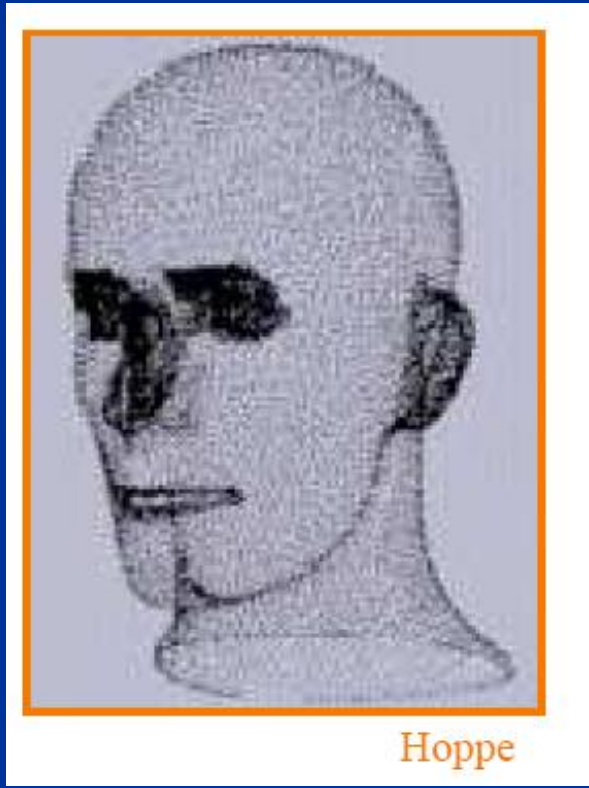
3D Object Representations





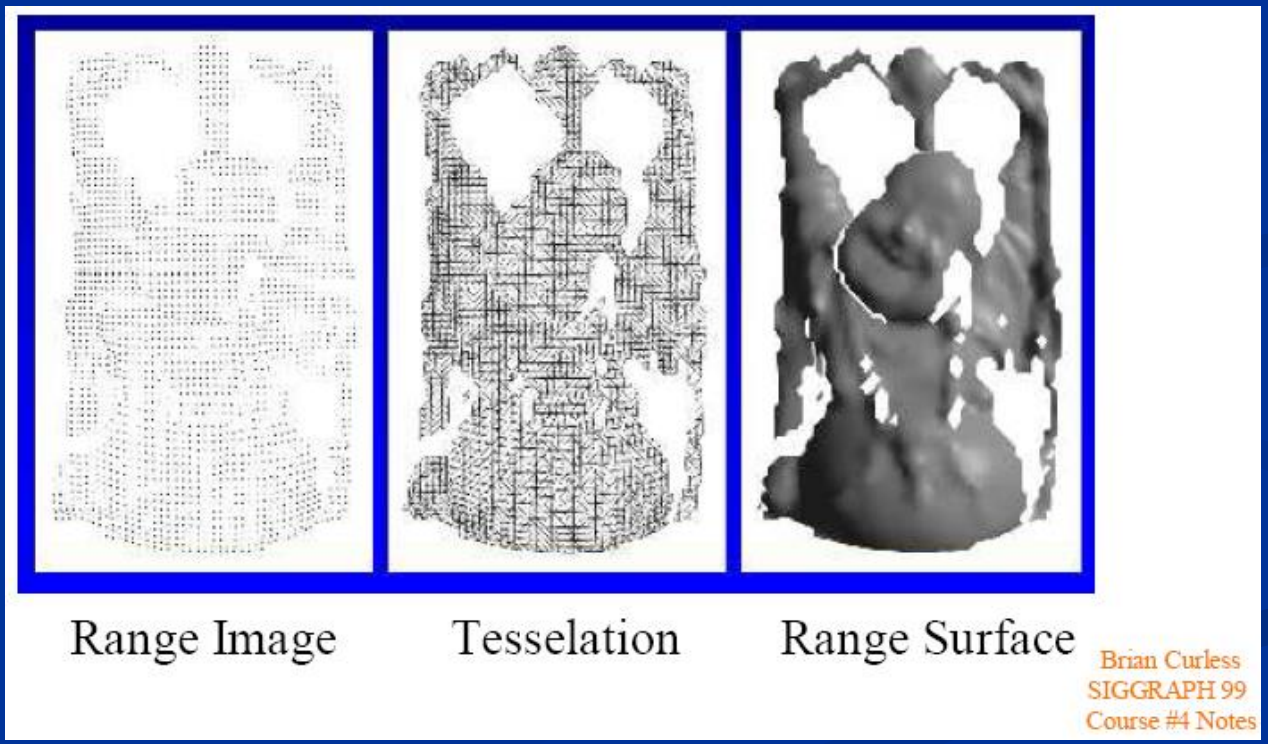
Point Cloud

- Sekumpulan titik pada ruang 3D
  - Didapat dari range finder, computer vision, dll



Range Image

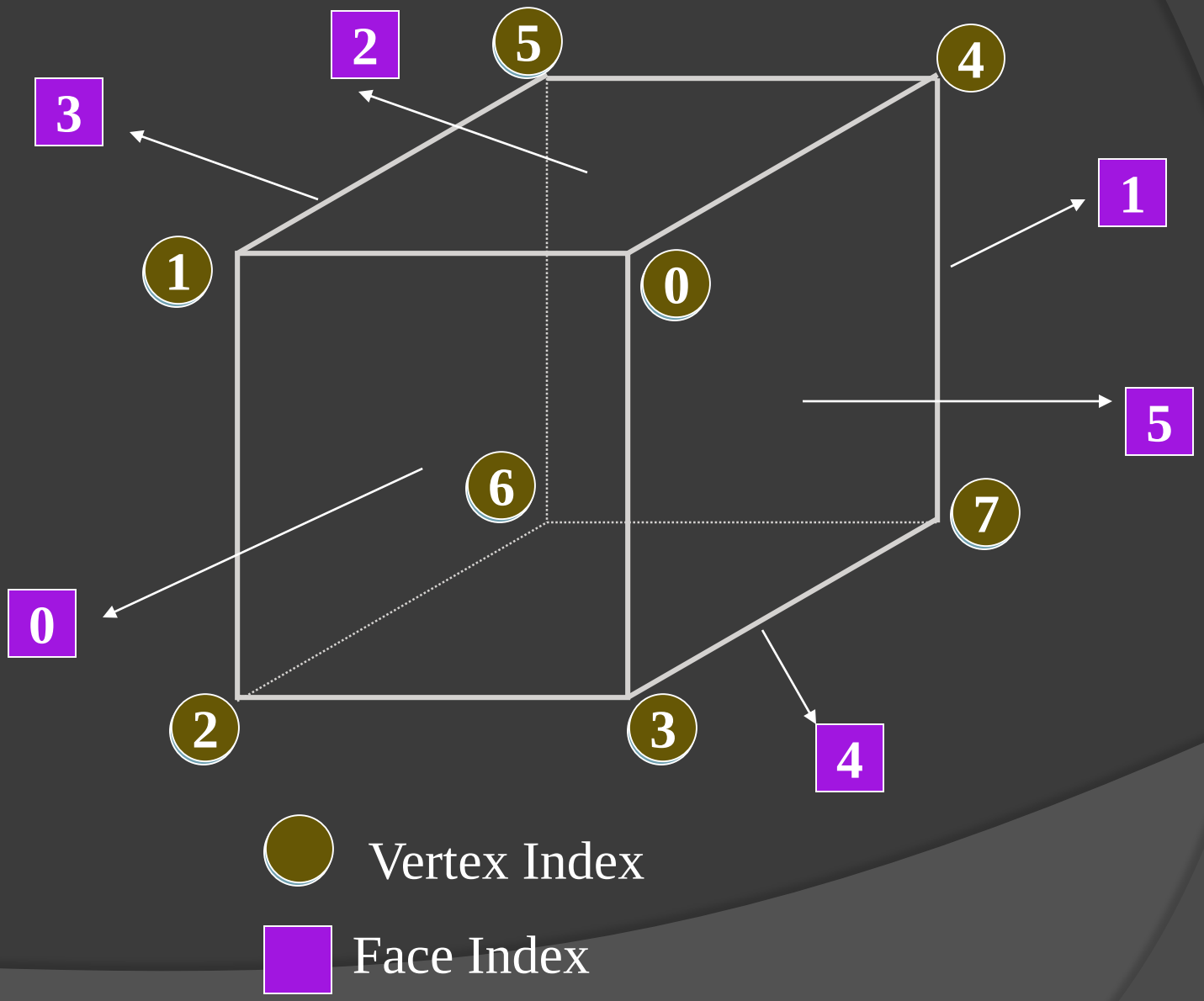
- Titik 3D yang menyatakan kedalaman
  - Didapat dari range scanner / 3D scanner



Polyhedron

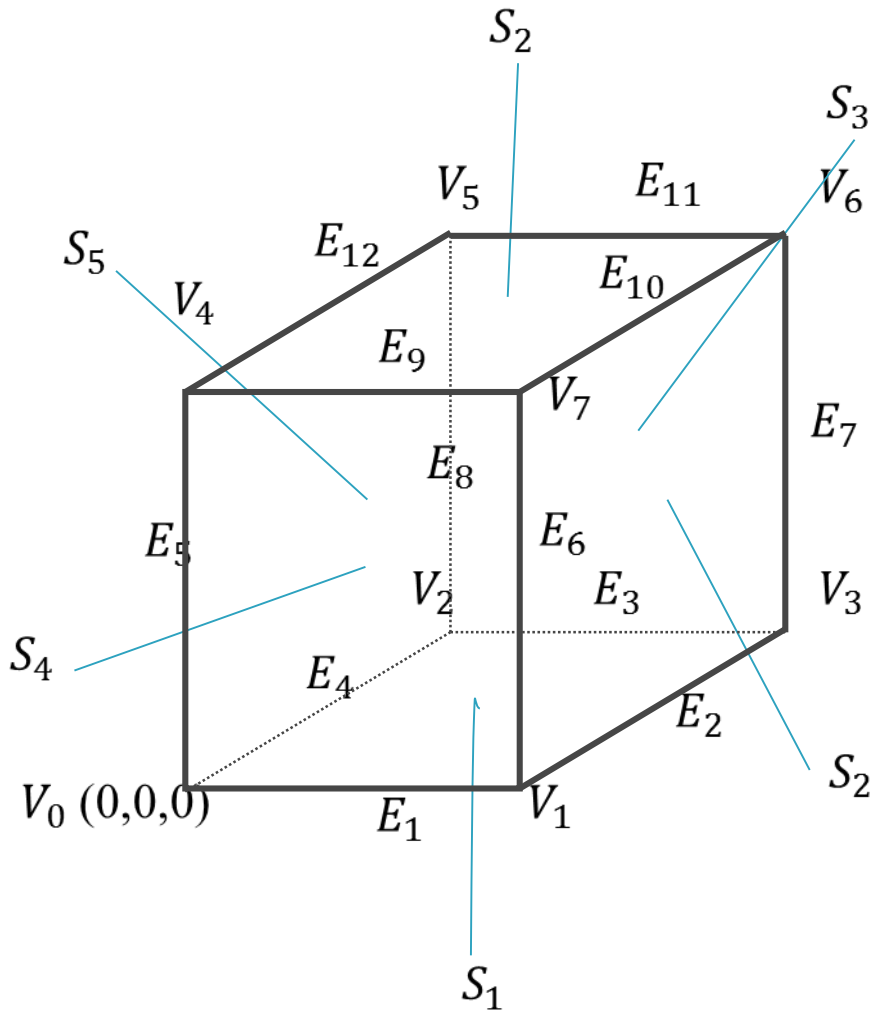
- gabungan dari sejumlah **terhingga (finite)** daerah-daerah bidang-segi, sedemikian, sehingga setiap sisi dari suatu daerah bidang-segi merupakan sebuah sisi dari tepat sebuah bidang-segi lain.
- Polyhedron adalah rangkaian jala polygon (polygon mesh) dengan kriteria sbb
  - Sedikitnya 3 edge bertemu pada setiap vertex.
  - Setiap edge dipakai oleh 2 faces
  - Faces tidak saling menembus, tetapi berhenti pada suatu edge.
- *Euler's formula* : If F, E, V represent the number of faces, vertices and edges of a polyhedron, then $V + F - E = 2$.

Vertices and Faces - E.g. Cube



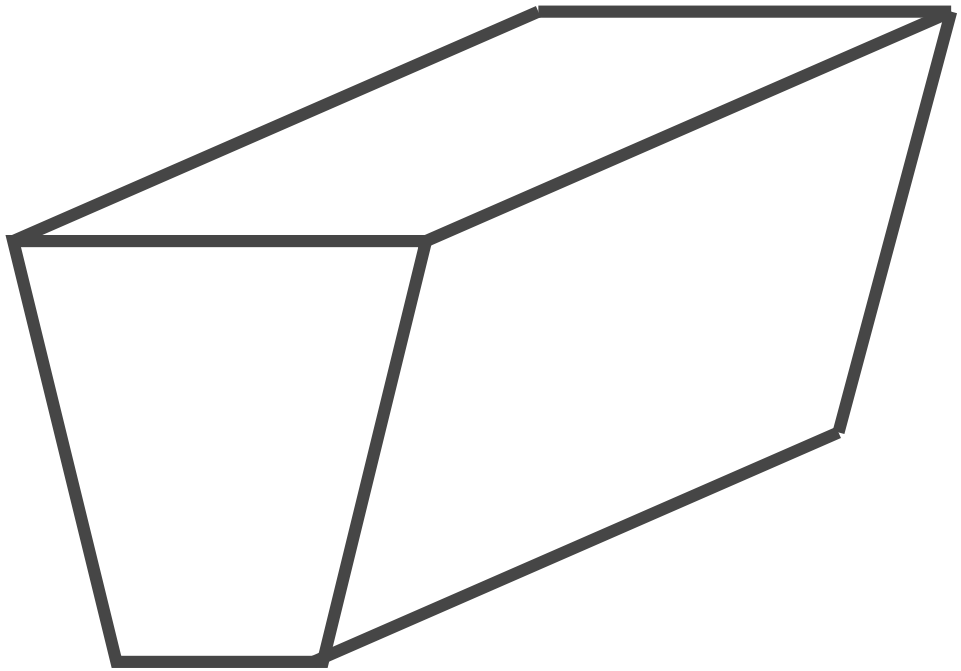
Contoh 1

- Jika diketahui sebuah objek 3D berbentuk kubus (polygon segi 4). Tentukan Vertex, edge dan surfacenya



Contoh 2

- Apakah objek berikut adalah polyhedron(jala polygon)?



Menggambar Polyhedron dalam Library OpenGL

- Ada 2 cara
  - Method1 : Fitting the surface with a polygon mesh. Membungkus permukaan objek polyhedron dengan susunan jala polygon. Proses ini disebut juga dengan surface tesselation
  - Method 2 : Memakai fungsi yang disediakan library GLUT

Method-1 Polygon Mesh

- Dalam membuat polygon ke dalam bentuk *surfaces,* tidak terbatas hanya menggunakan GL_POLYGON
- Dapat juga mengunakan fungsi-funsgi berikut:
  - GL_TRIANGLES
  - GL_TRIANGLE_STRIP
  - GL_TRIANGLE_FAN
  - GL_QUADS
  - GL_QUAD_STRIP
- Bagaimana penggunan GL_TRIANGLES untuk kasus pada contoh 1?

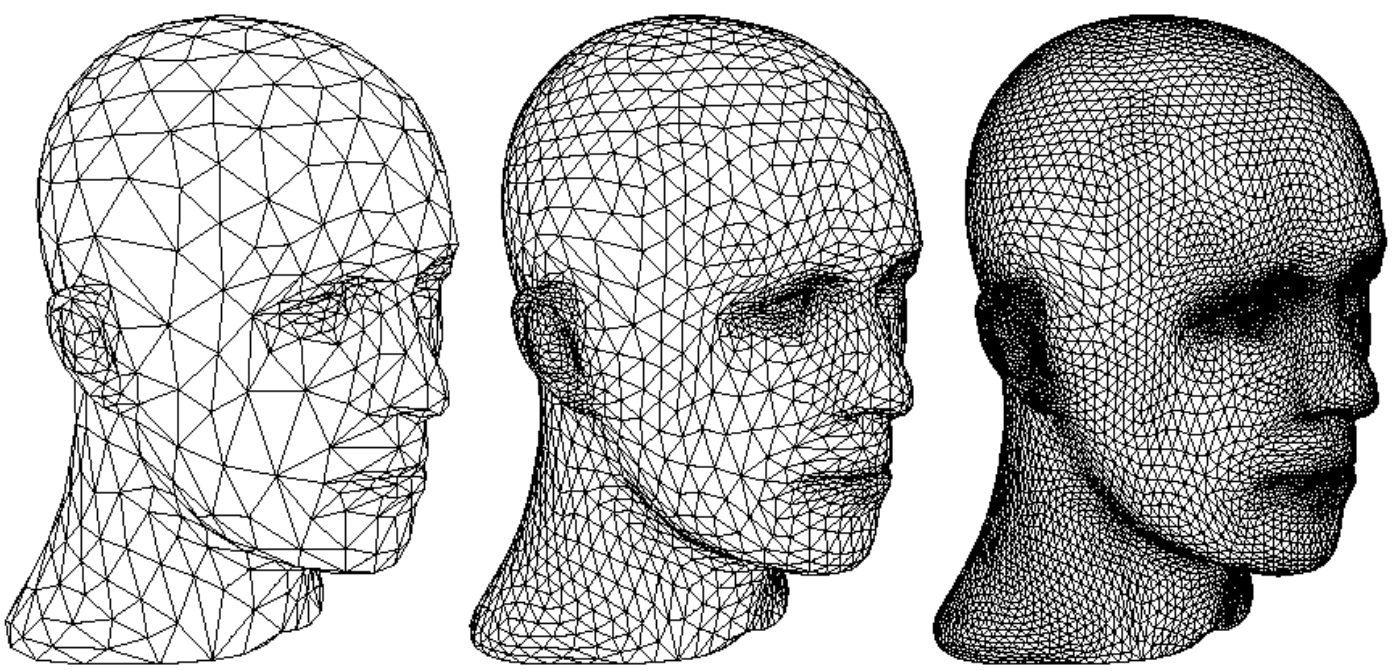
Draw Polygon Mesh in OpenGL (dengan vektor normal)

```
glBegin(GL_TRIANGLES); // draw a cube with 12 triangles polygon
// front surface ================
glVertex3fv(v0); // v0-v1-v7
glVertex3fv(v1);
glVertex3fv(v7);
glVertex3fv(v7); // v7-v4-v0
glVertex3fv(v4);
glVertex3fv(v0);
// right surface ================
glVertex3fv(v1); // v1-v3-v6
glVertex3fv(v3);
glVertex3fv(v6);
glVertex3fv(v6); // v6-v7-v1
glVertex3fv(v7);
glVertex3fv(v1);
// draw other 4 faces
glEnd();
```

Catatan:
Untuk mereduksi jumlah penganggilan funsi dan penggunaan yang redundan, dapat menggunakan **vertex arrays** ke dalam 3 bentuk funsi open GL yaitu: **glDrawArrays()**, **glDrawElements()** and **glDrawRangeElements()**.
Atau pendekatan yang lebih baik menggunkan vertex buffer objects (VBO) or display lists

Polygon Mesh

- Polygon mesh ini juga bisa dipakai untuk memodelkan permukaan objek lainnya



Method 2- OpenGL Polyhedron Functions

- Fungsi menghasilkan wire frames dimana dapat digunakan secara langsung dan mudah untuk *cube* yang dibentuk oleh regular polygon seperti: Tetrahedron, Octahedron, Hexahedron, Dodecahedron, atau Icosahedron

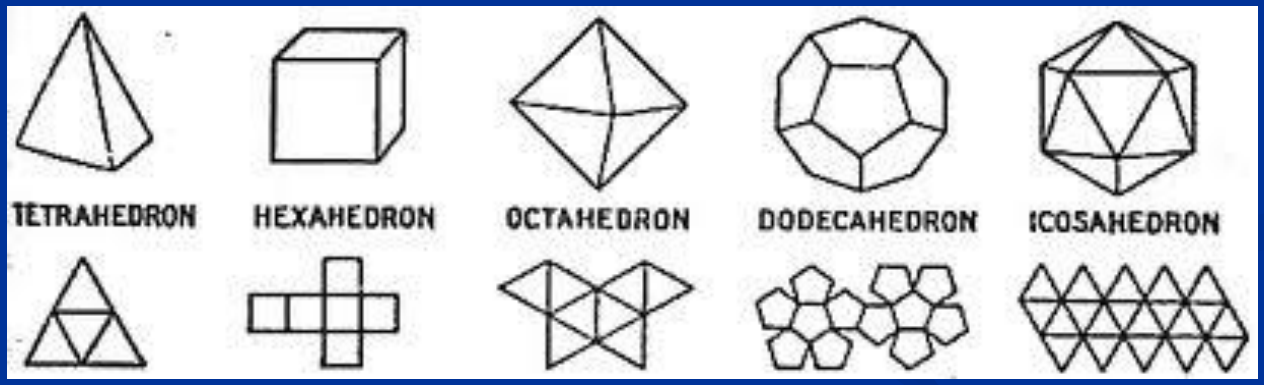
  Contoh: glutWireX(), X merupakan nama cube

- Fungsi menghasilkan "polyhedra facets as shaded fill areas" - dengan karakteristik fungsi dideterminasi oleh properti lighting dan material

  Ex: glutSolidX(), X merupakan nama cube

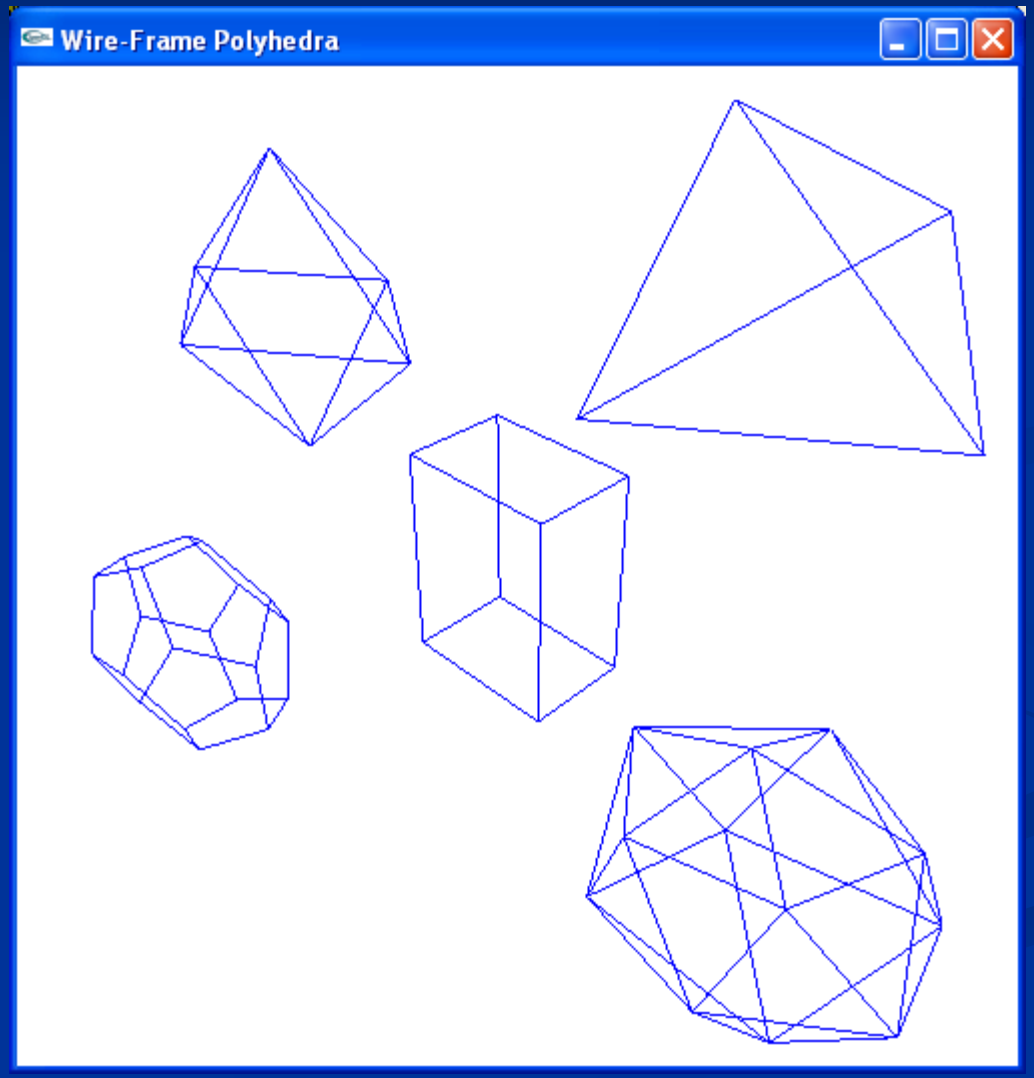
Regular Polyhedra (Platonic Solids)

- Jika semua face pada polyhedron adalah identik dan berupa regular polygon, maka polyhedron tsb disebut *platonic solid*.
- Hanya ada 5 jenis platonic solid

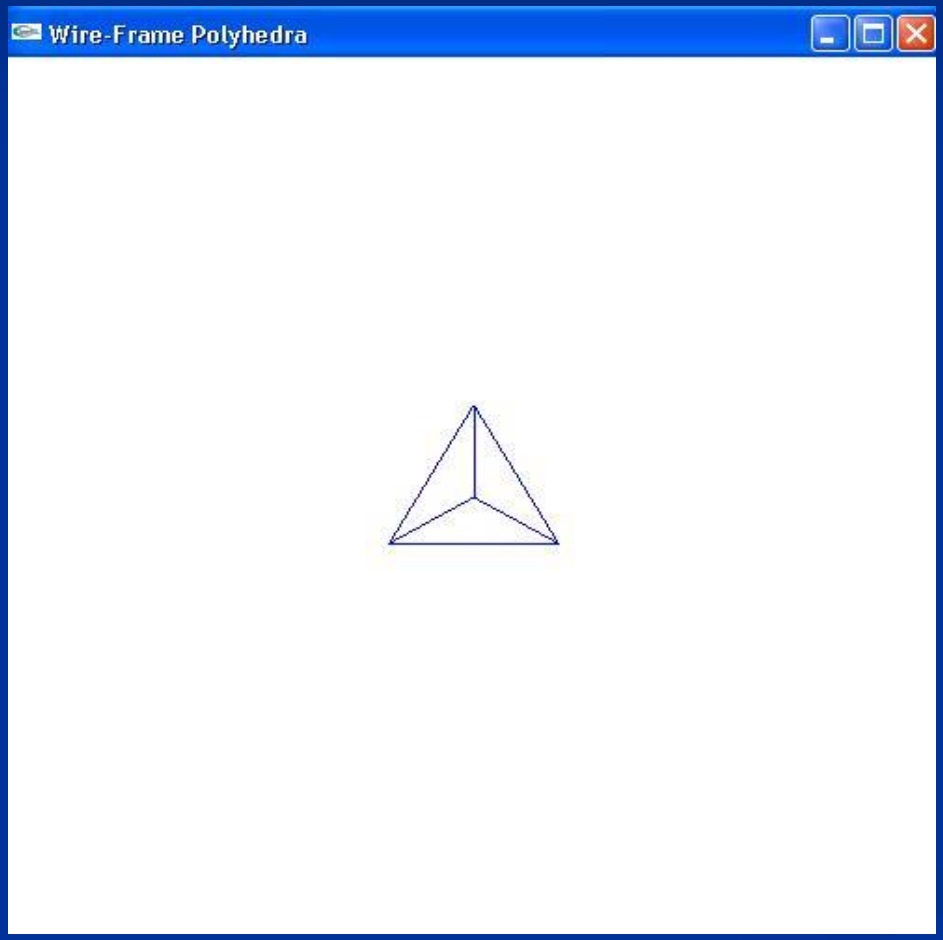


  - Tetrahedron (or triangular pyramid) has 4 faces
  - Hexahedron (or cube) with 6 faces
  - Octahedron with 8 faces
  - Dodecahedron with 12 faces
  - Icosahedron with 20 faces

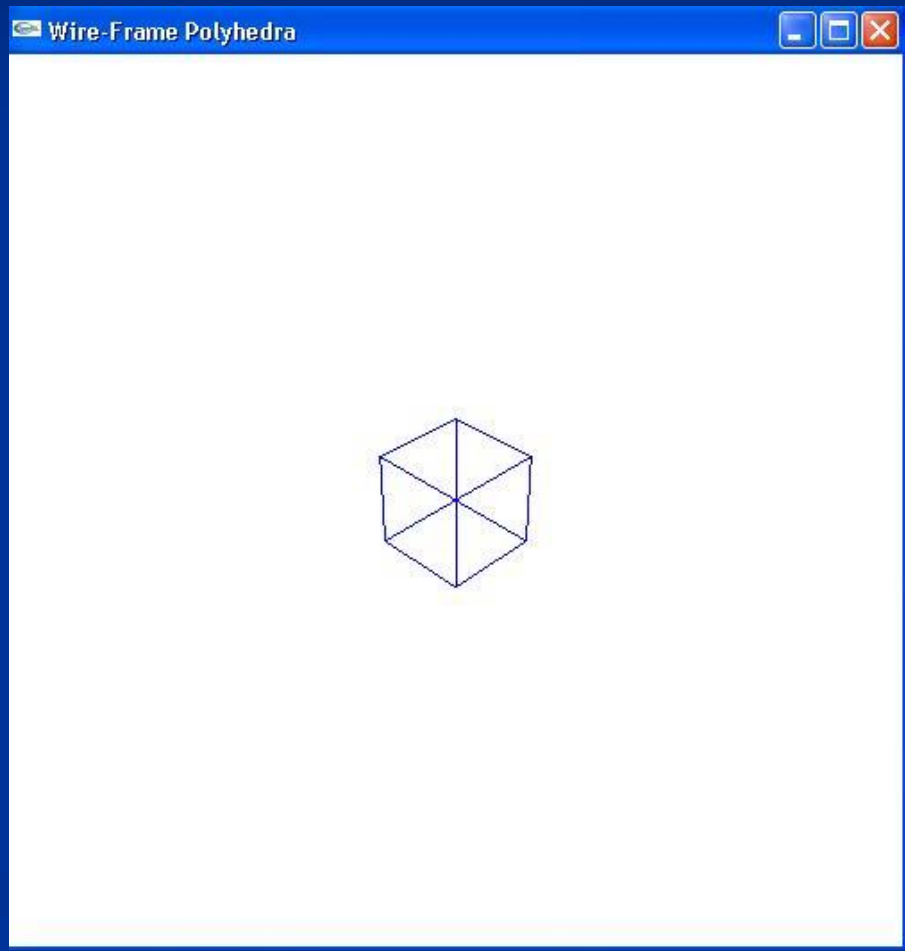
GLUT Library of Polyhedron Functions



glutWireTetrahedron() and glutWireCube(1.0)

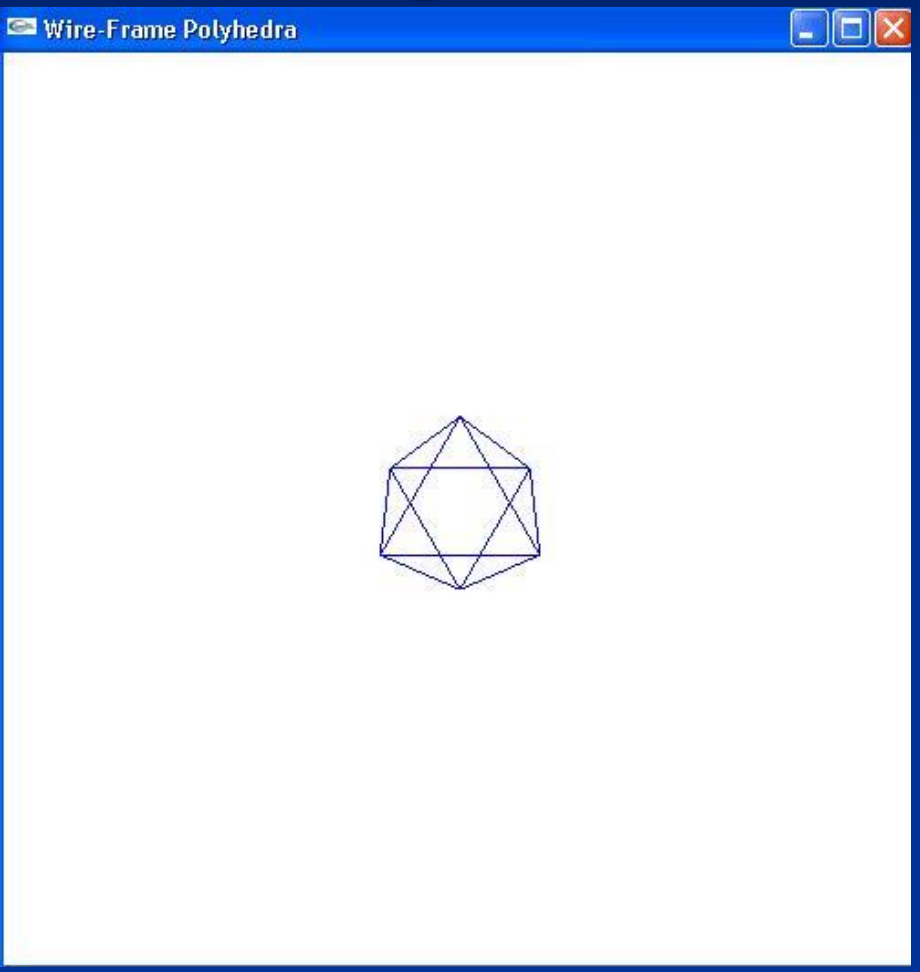


4 faces

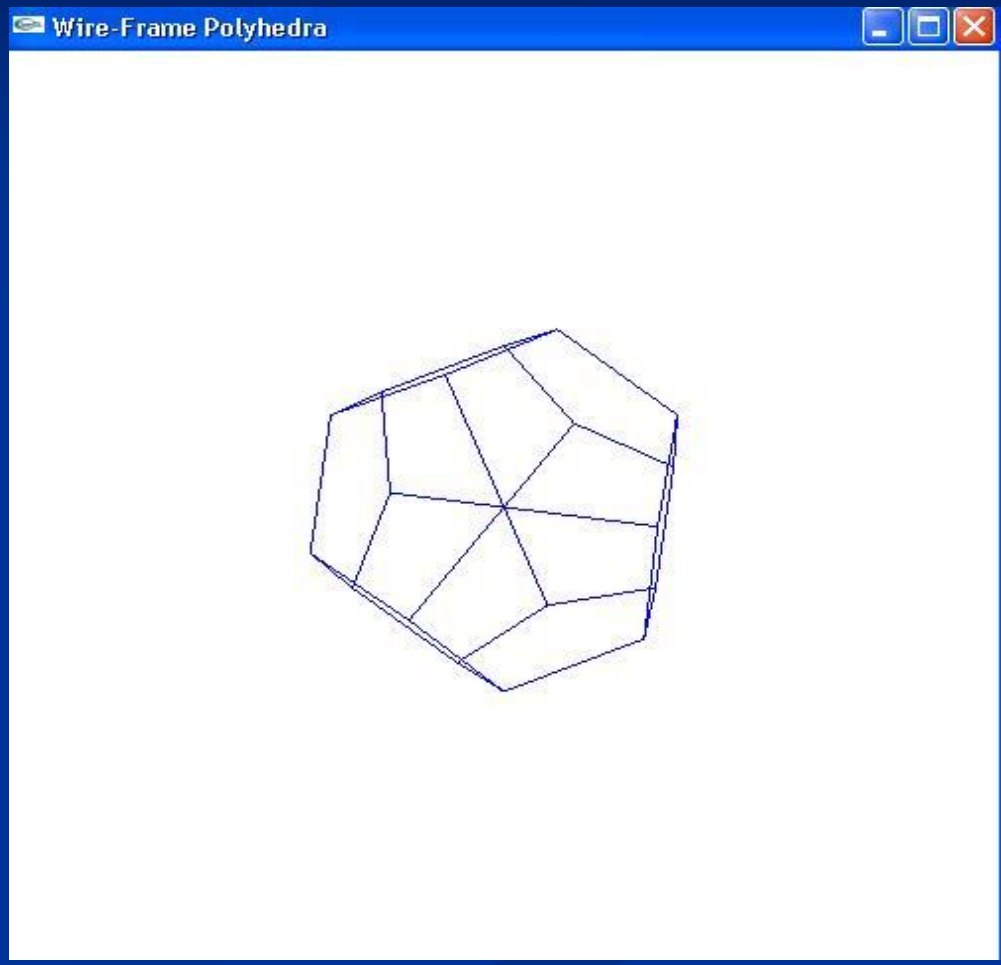


6 faces

glutWireOctahedron() and glutWireDodecahedron()

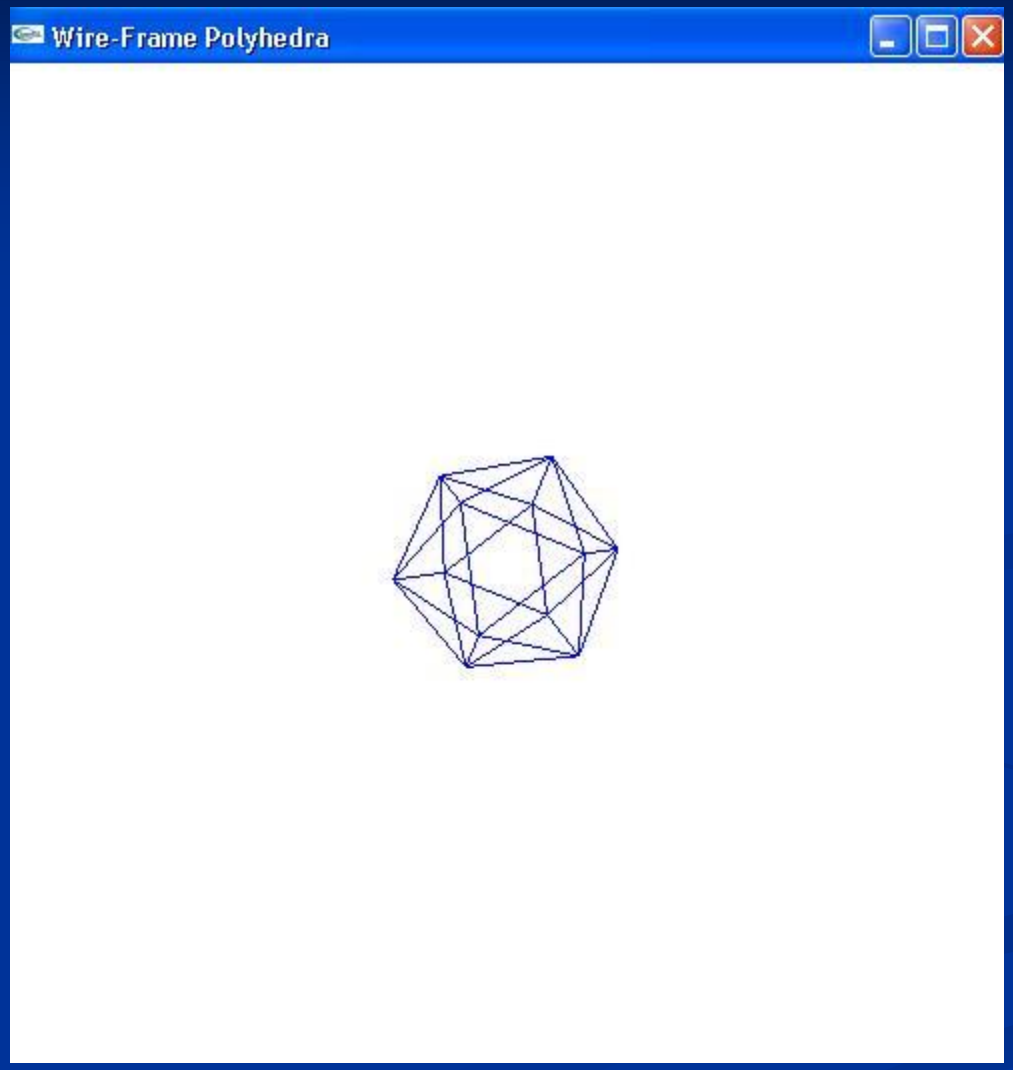


8 faces



12 faces

And, glutWireIcosahedron()



20 faces

Quadrics

- **Objek yang didefinisikan sebagai persamaan quadratics**
  - Sphere

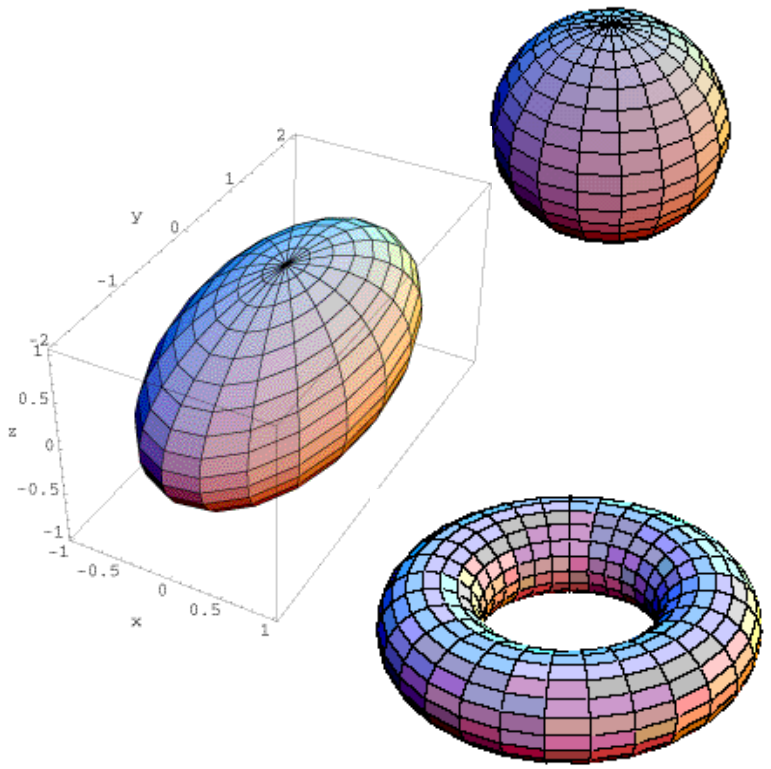
$$x^2 + y^2 + z^2 = r^2$$

  - Ellipsoid

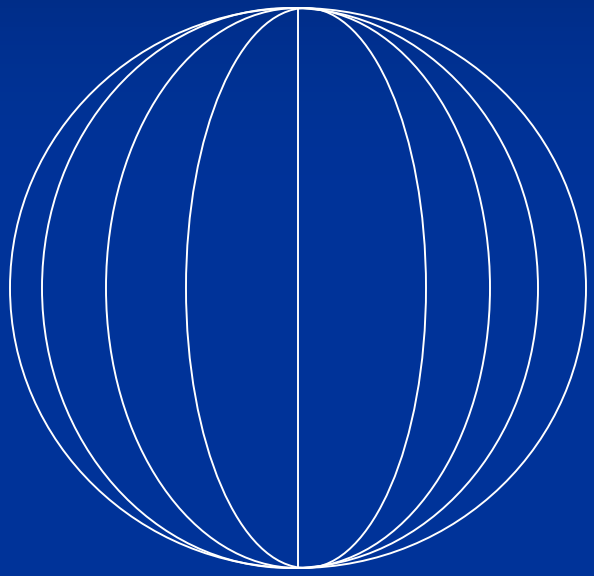
$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

  - Torus

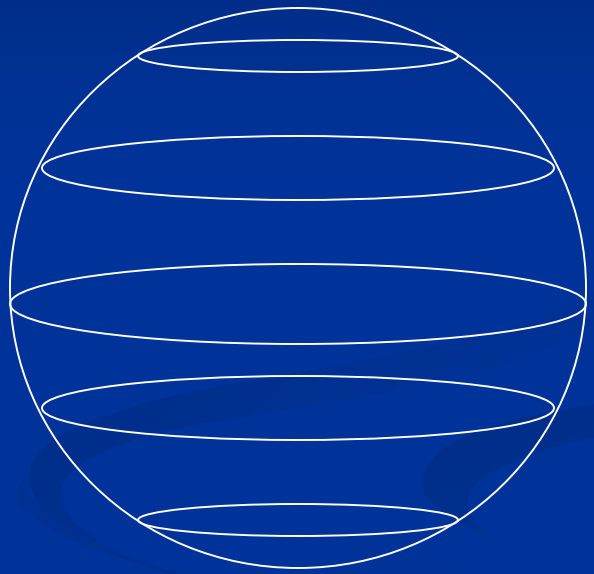
$$\left[r - \sqrt{\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2}\right]^2 + \left(\frac{z}{r_z}\right)^2 = 1$$



GLUT Sphere



Longitudes

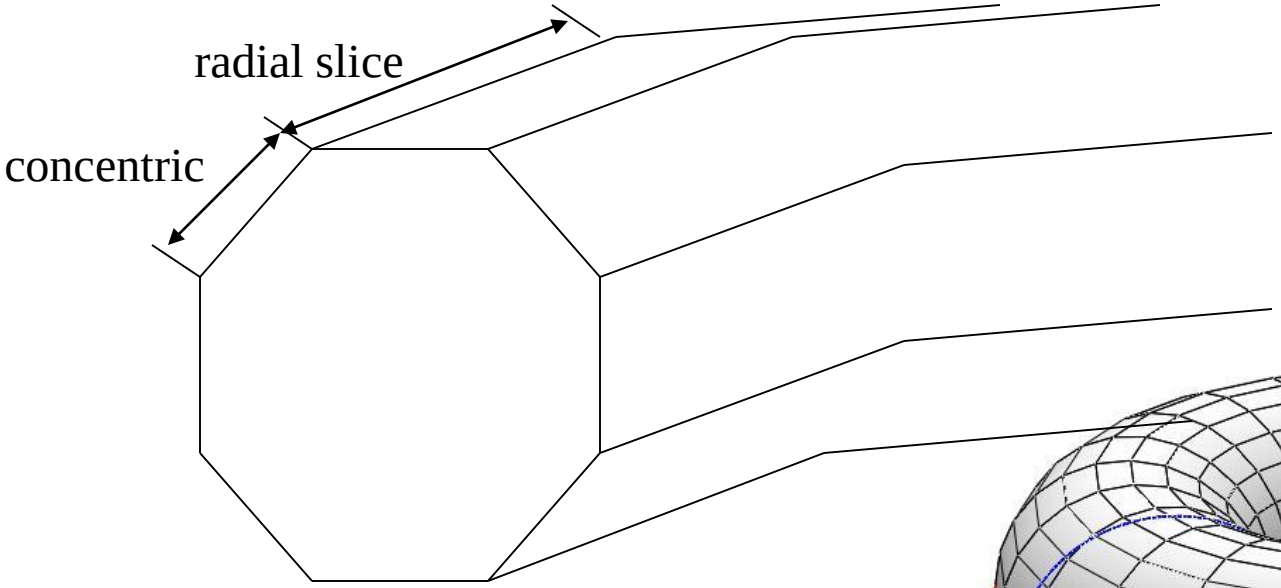


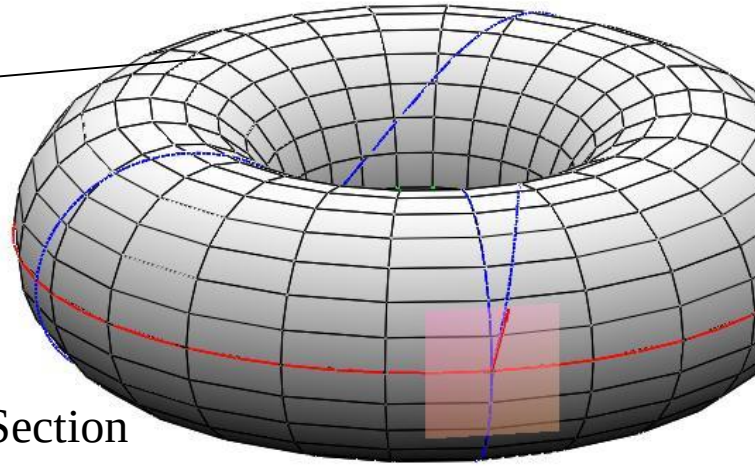
Latitudes

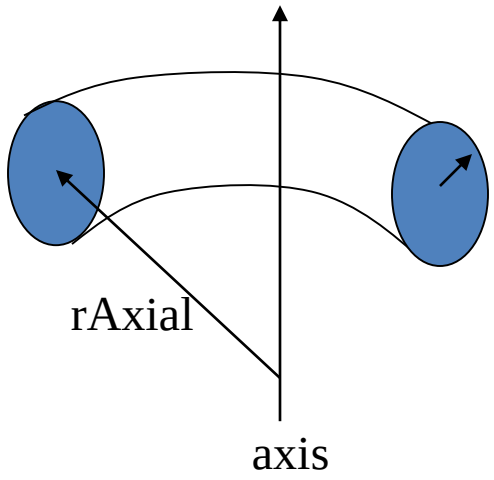
glutSolidSphere(r, nLongitudes, nLatitudes);

GLUT Torus

glutSolidTorus(rCrossSection, rAxial, nConcentrics, nRadialSlices);

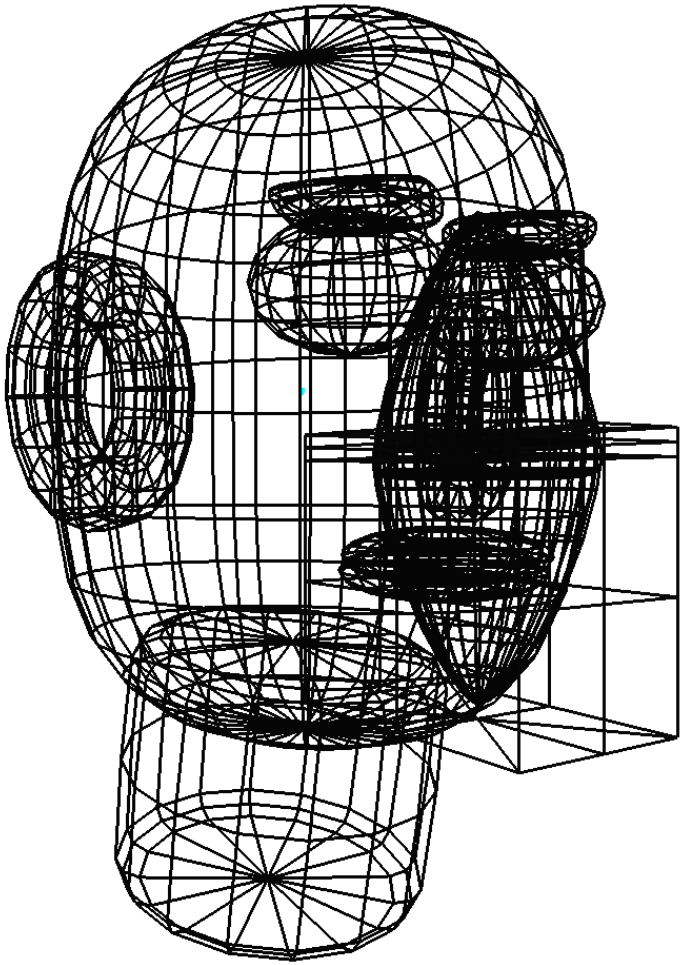


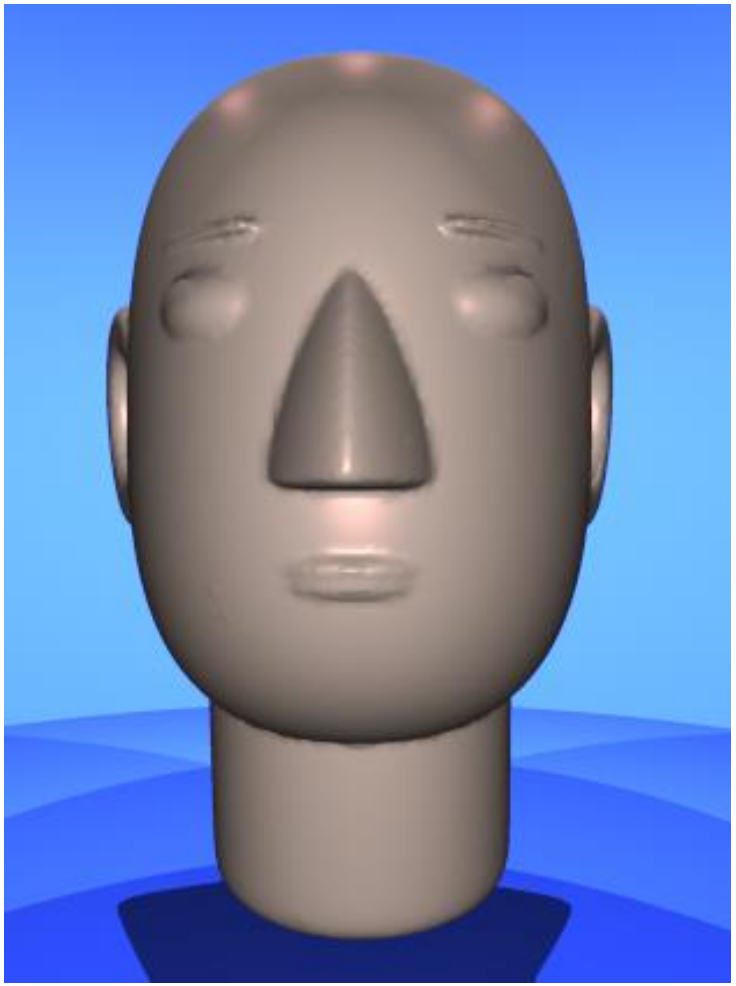




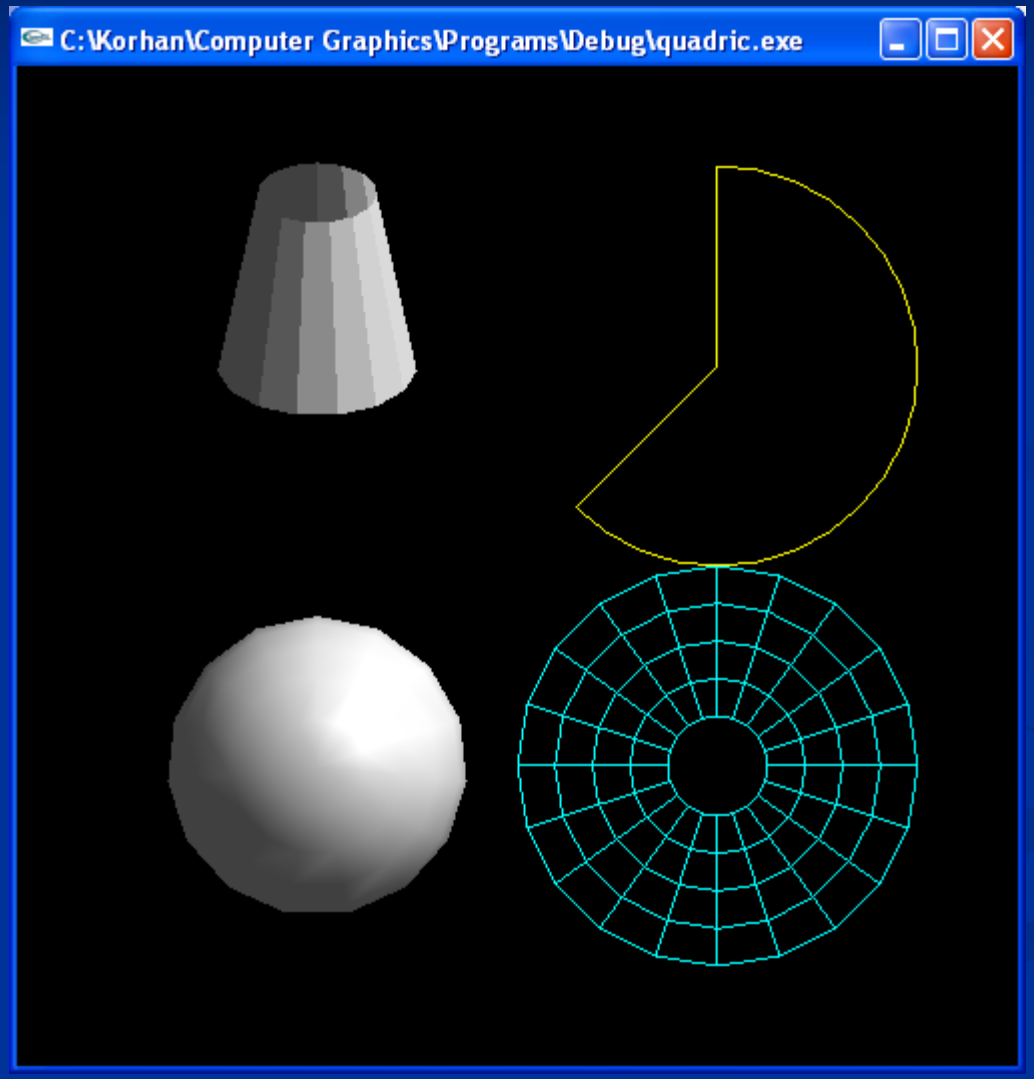
rCrossSection = inner radi
rAxial = outer radius

Object with Quadrics



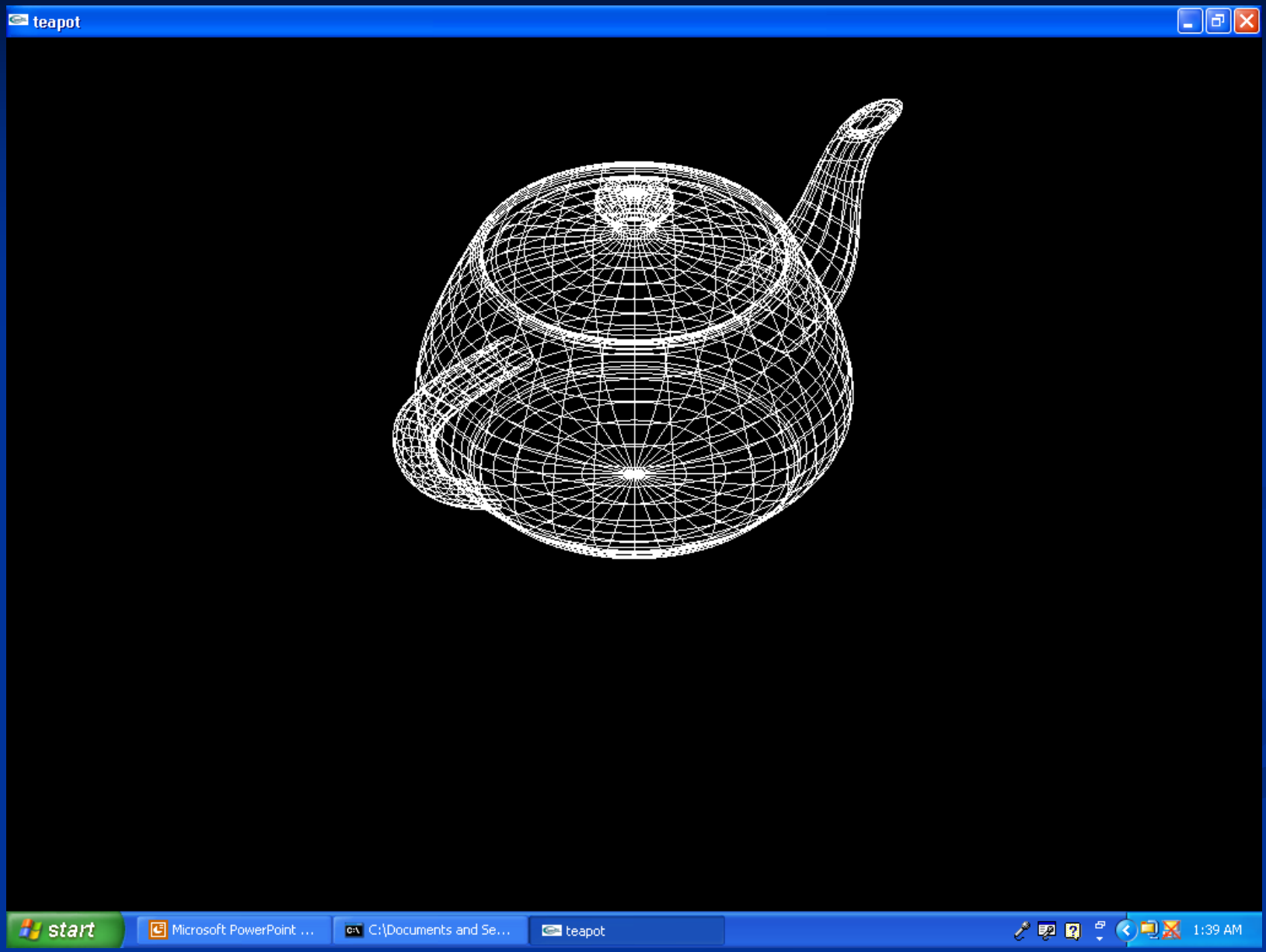


GLU Quadric-Surface Functions

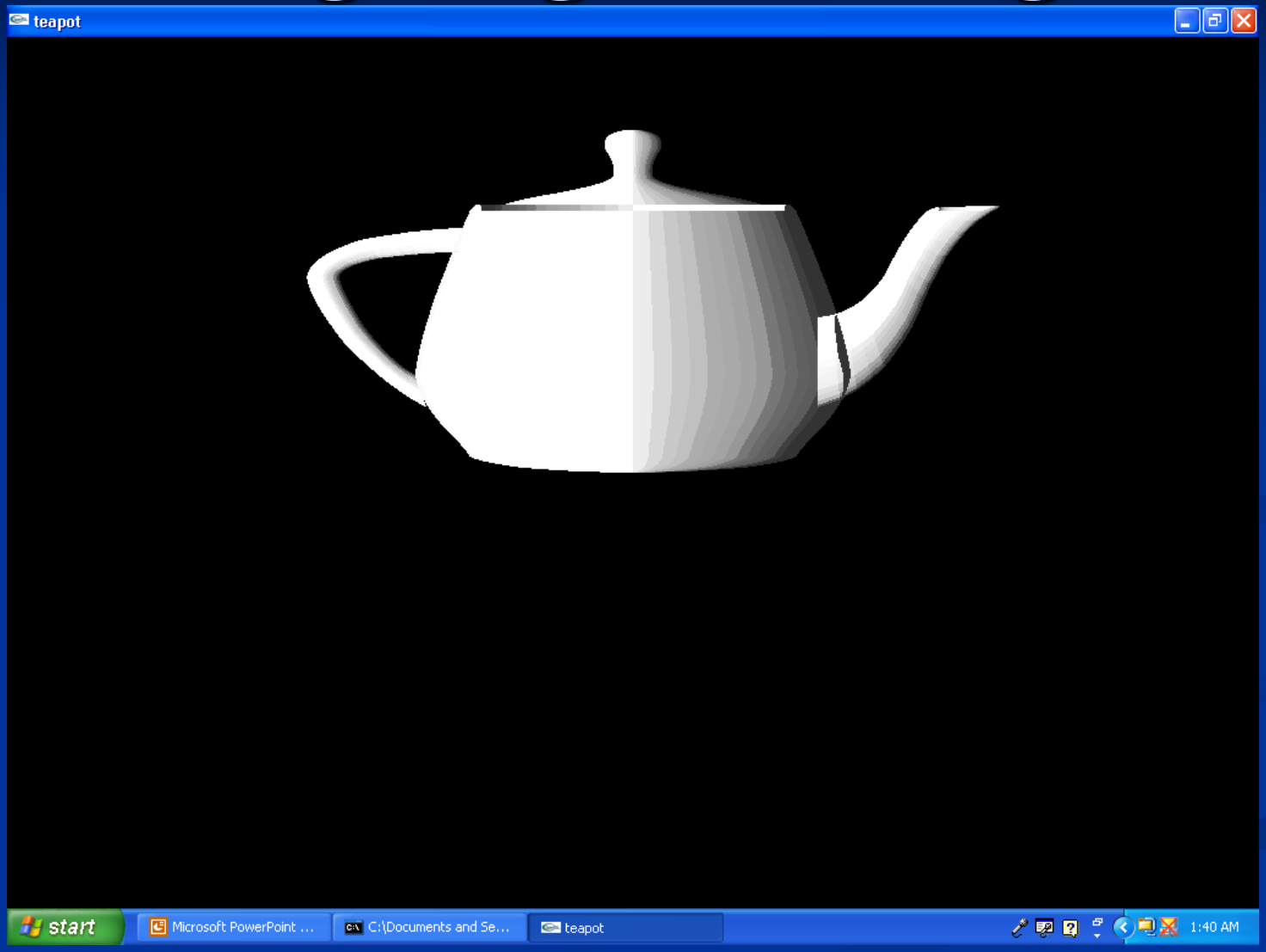


Quadric.c

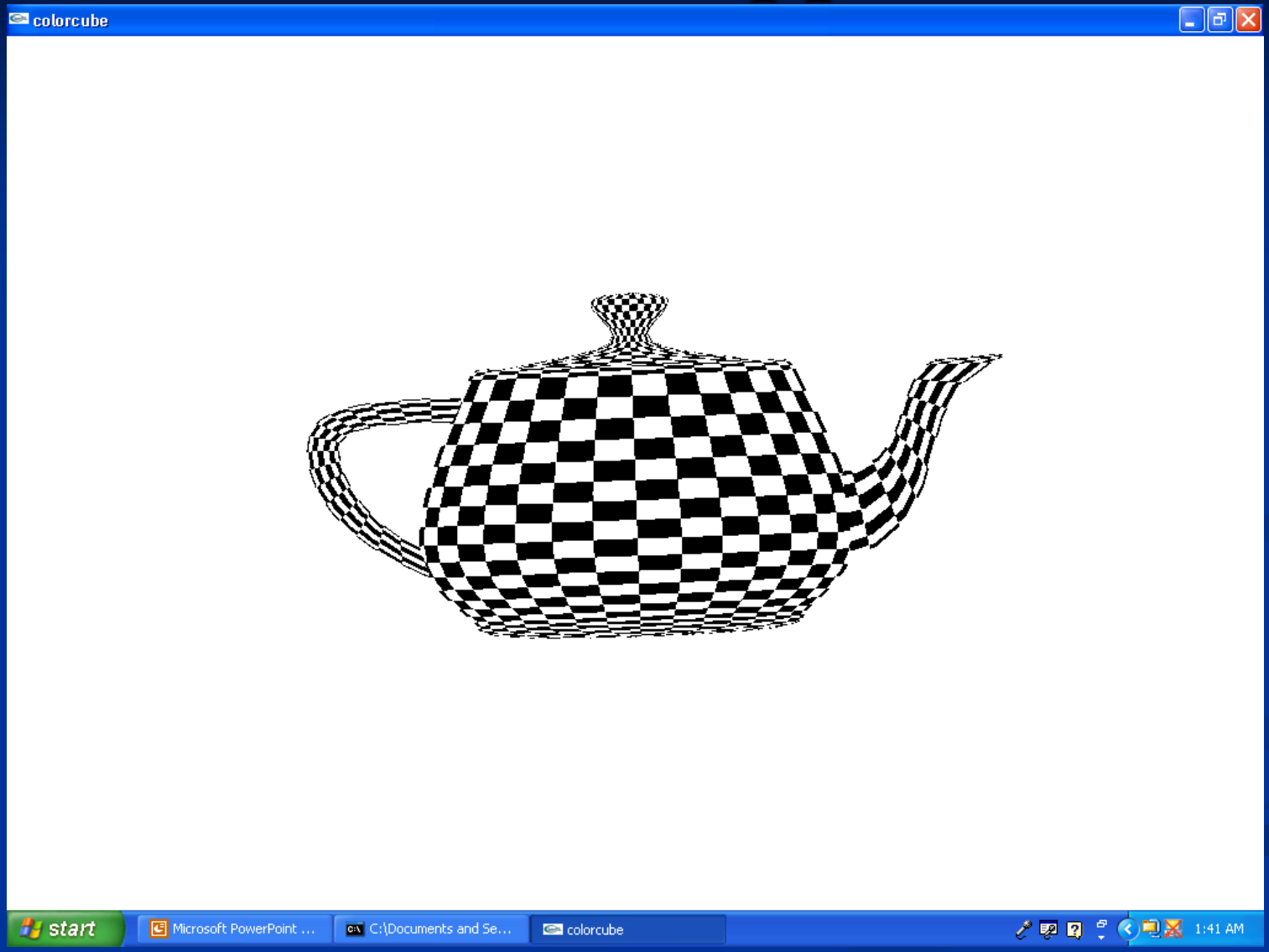
wireframe


teapot
start
Microsoft PowerPoint ...
C:\Documents and Se...
teapot
1:39 AM

Lighting & shading



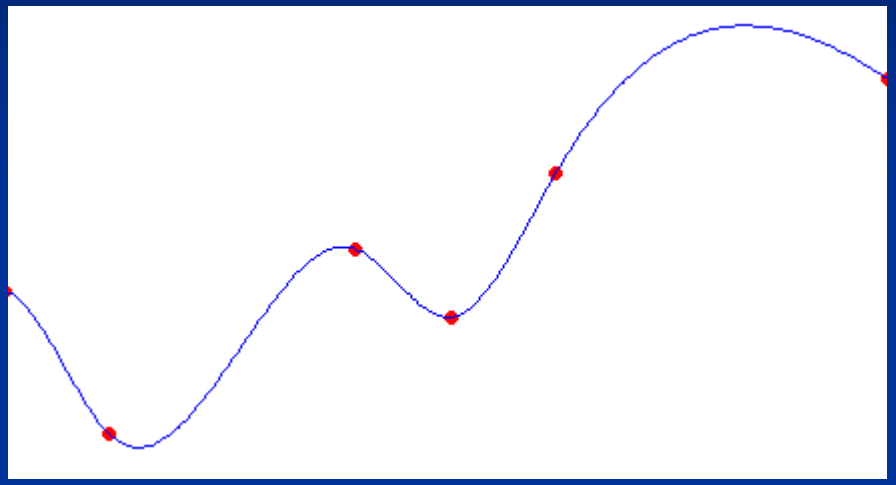
Texture mapped



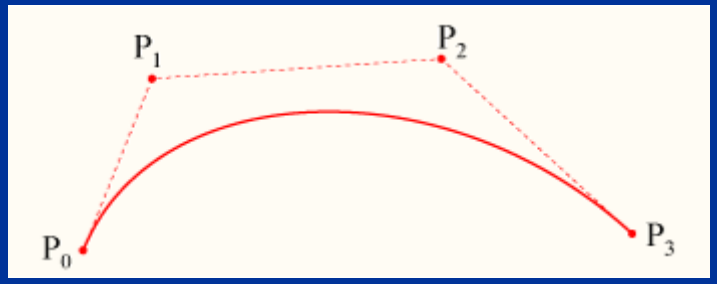
C. Spline Representations

- Splines curves digunakan untuk mendesain lengkungan pada permukaan berbasis pada sekumpulan (himpunan) titik pengontrol kurva
- Tititk Pengontrol (Control points)
  - Himpunan titik koordinat yang mengontrol bentuk kurva
- Interpolasi (Interpolation)
  - Semua control points tersambung satu sama lain pada garis kurva
- Aproksimasi (Approximate)
  - Semua atau beberapa control points terletak di luar garis kurva

Spline Representations

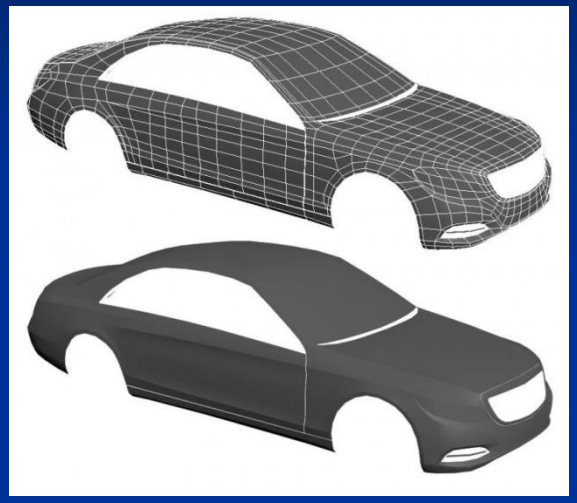


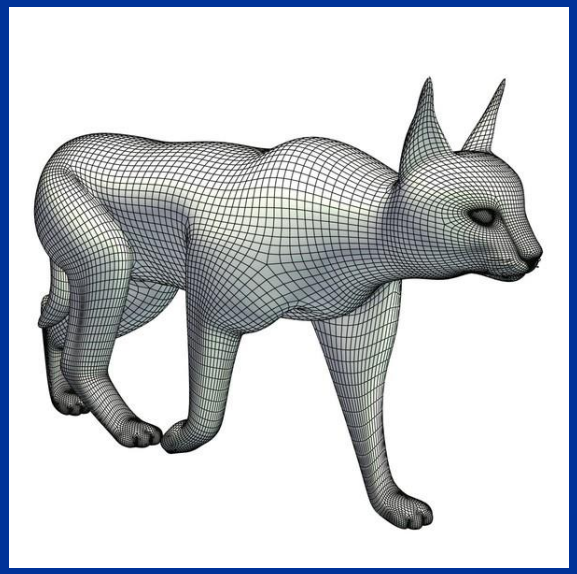
Interpolated

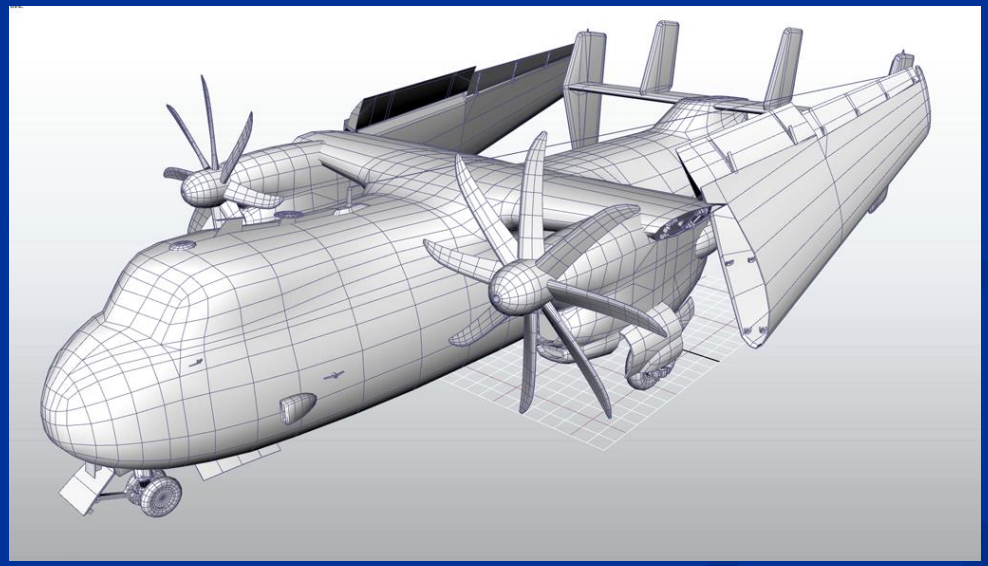


Approximate

Spline Representation

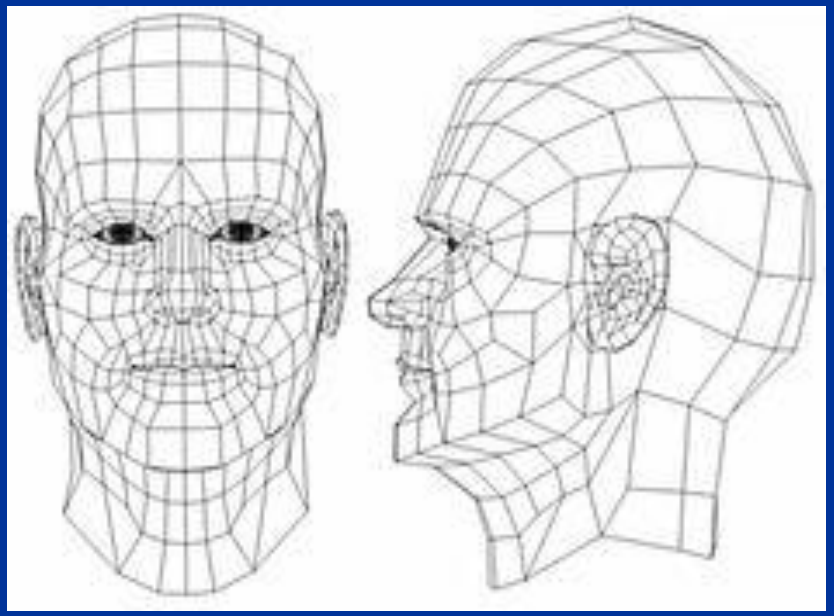


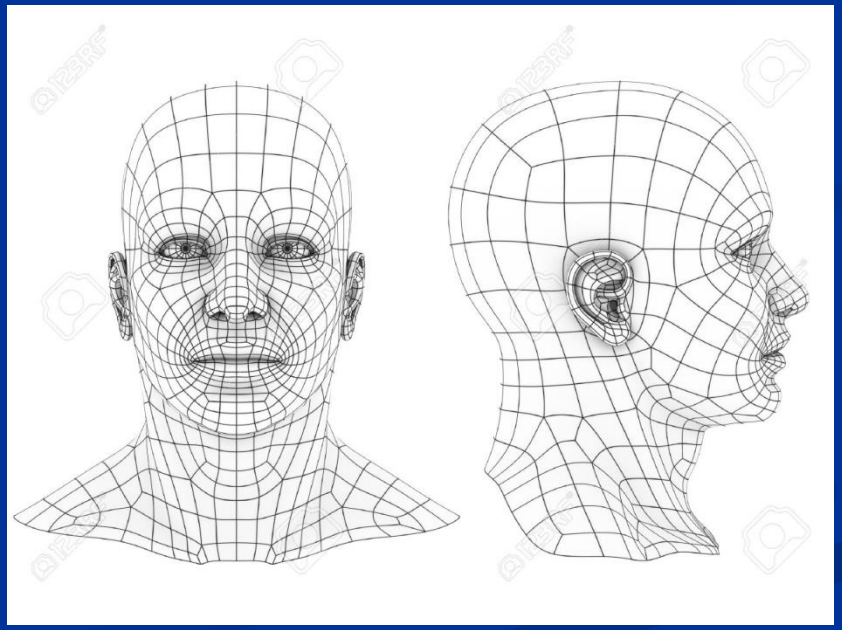




Spline Representation

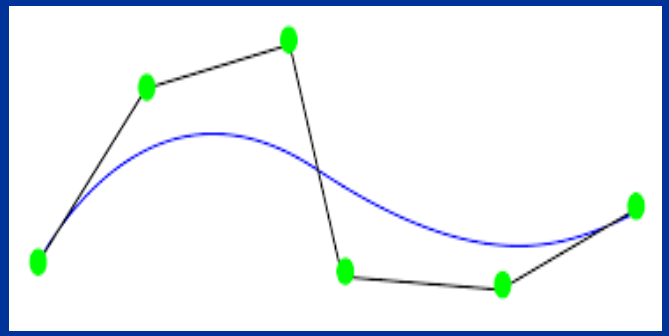
- Perbedaan antara penggambaran Polygon dan Polygon with spline curve

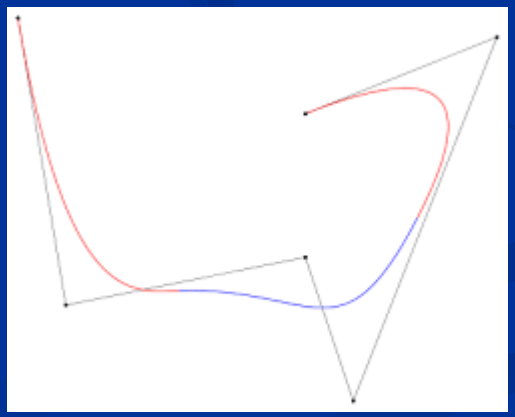




Bezier Spline Curves

- Developed by French engineer Pierre Bézier for use in the design of Renault automobile bodies
- Easy to implement
  - Widely used in CAD systems, graphics, drawing and painting packages



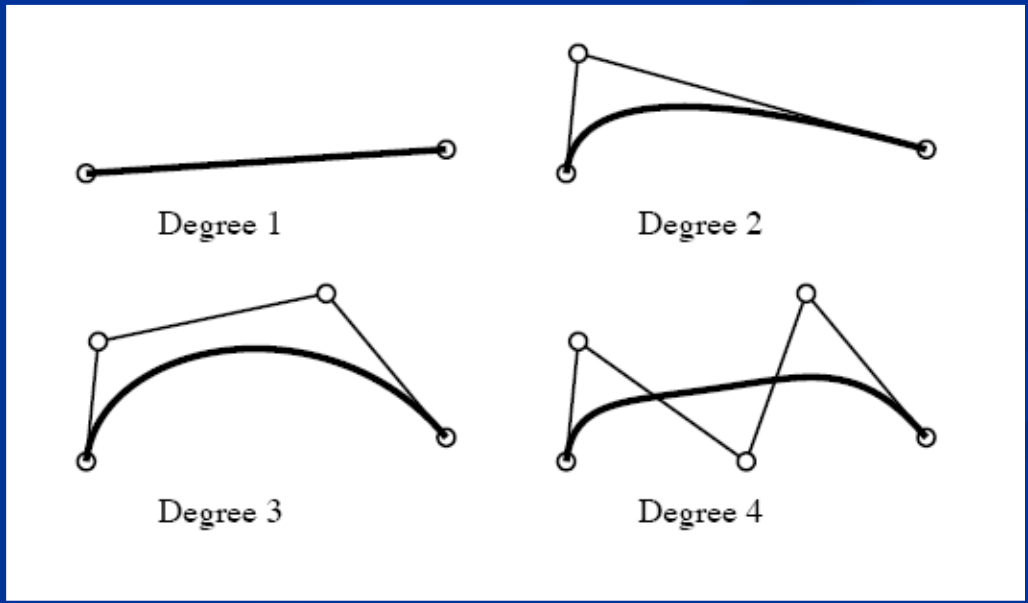


Bezier Curve Equations

- Diketahui sejumlah n +1 control points,

  $p_k = (x_k, y_k, z_k)$

  nilai k antara 0 sampai n
- Persamaan garis Bezier akan membentuk titik-titik garis kurva sesuai control point yang didefinisikan



Bezier Curve Equations

- Degree 1 – Linear Curve

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , t \in [0,1]$$

- Degree 2

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \ , t \in [0,1].$$

- Degree 3

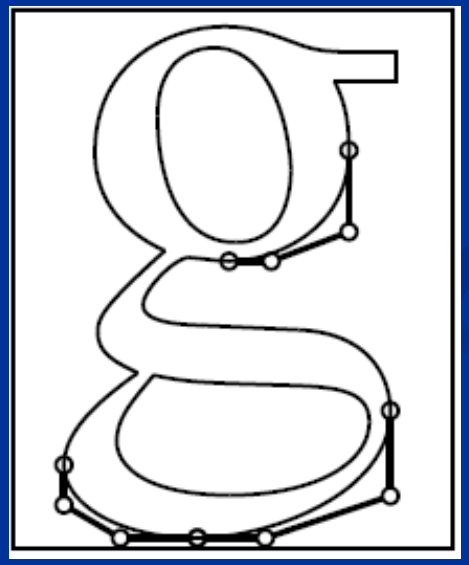
$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , t \in [0,1].$$

- Degree n

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n\mathbf{P}_0 + \binom{n}{1}(1-t)^{n-1}t\mathbf{P}_1 + \cdots \\ &\quad \cdots + \binom{n}{n-1}(1-t)t^{n-1}\mathbf{P}_{n-1} + t^n\mathbf{P}_n, \quad t \in [0,1],\end{aligned}$$

Bezier Spline Curves

- A common use for Bezier curves is in font definition



Contoh Bezier Curve

- Diberikan suatu kurva bezier derajat 1 dengan titik awal (-4,4) dan titik akhir (0,0).
- Buatlah plotting titik pada kurva sebanyak 6 titik (termasuk titik awal dan akhir).

Degree 1

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , \ t \in [0,1]$$

- P0=(-4,4) P1=(0,0)

- k=5; step=1/k=1/5=0.2
- t=0 → (-4,4) + 0(4,-4) =(4.-4)
- t=0.2 → (-4,4) + 0.2(4,-4) =(-3.2, 3.2)
- t=0.4 → (-4,4) + 0.4(4,-4) =(-2.4, 2.4)
- t=0.6 → (-4,4) + 0.6(4,-4) =(-1.6, 1.6)
- t=0.8 →(-4,4) + 0.8(4,-4) =(-0.8, 0.8)
- t=1 → (-4,4) + (4,-4) =(0,0)

OpenGL Approximation Spline Functions

- Bezier splines and B-splines can be displayed using OpenGL functions
- The core library contains Bezier functions, and GLU has B-spline functions
- Bezier functions are often hardware implemented

OpenGL Bezier-Spline Curve Functions

- We specify parameters and activate the routines for Bezier-curve display with
  - glMap1*(GL_MAP1_VERTEX_3, uMin, uMax, stride, nPts, *ctrlPts);
  - glEnable(GL_MAP1_VERTEX_3);
- and deactivate with
  - glDisable(GL_MAP1_VERTEX_3);
  - uMin and uMax are typically 0 and 1.0
  - stride=3 for 3D
  - nPts is the number of control points
  - ctrlPts is the array of control points

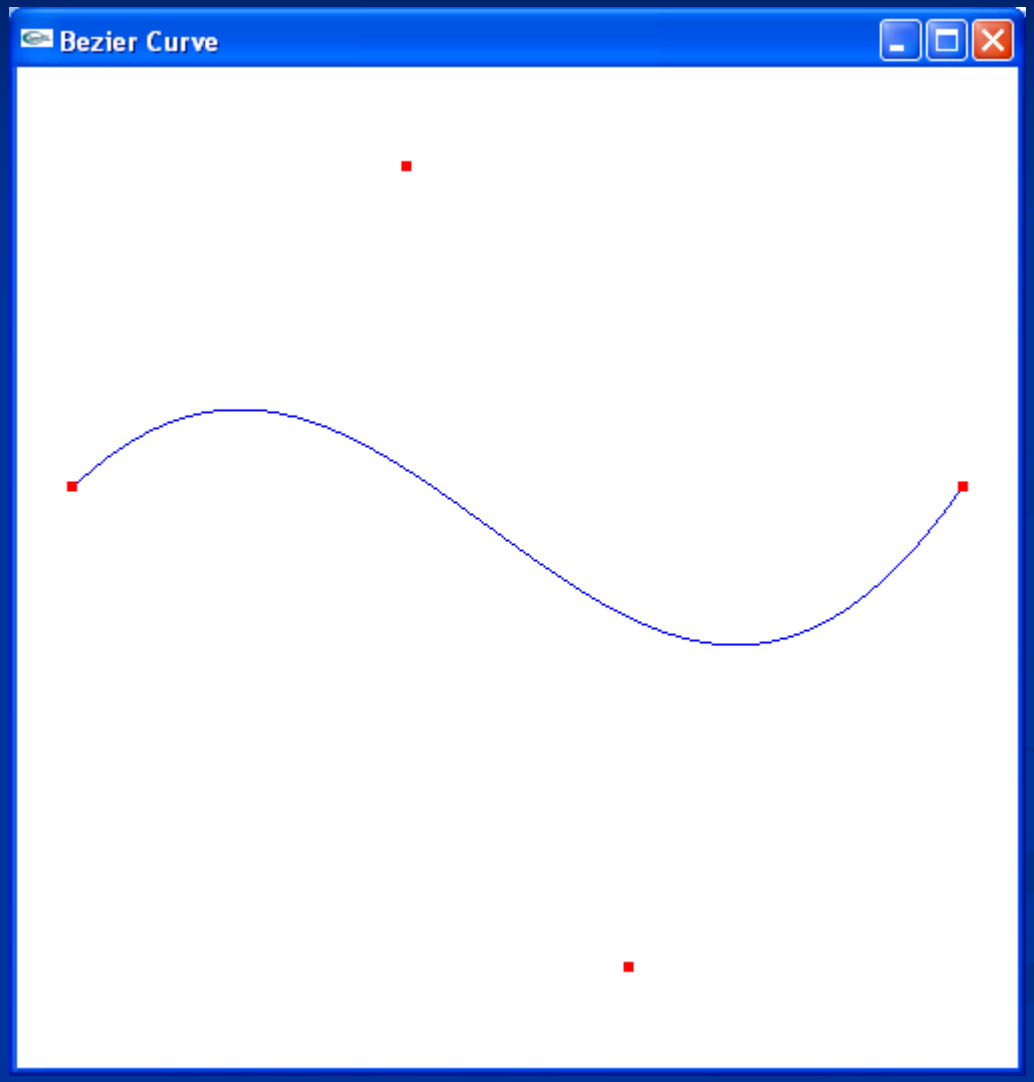
OpenGL Bezier-Spline Curve Functions

- After setting parameters, we need to evaluate positions along the spline path and display the resulting curve. To calculate coordinate positions we use

  glEvalCoord1*(uValue);

  where uValue is assigned some value in the interval from uMin to uMax

Example OpenGL Code



prog8OGLBezierCurve.cpp

Bezier Surfaces

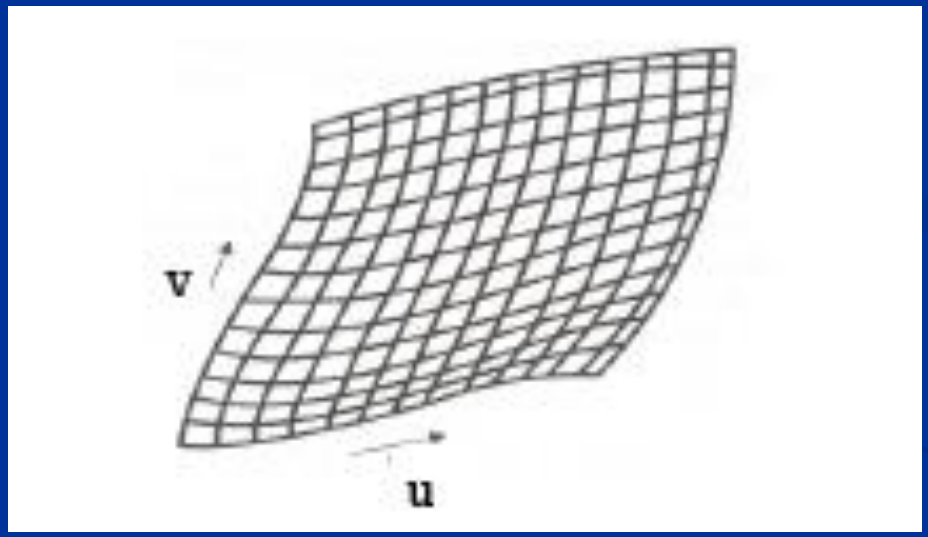
- Two sets of Bezier curves can be used to design an object surface

$$P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} p_{j,k} BEZ_{j,m}(v) BEZ_{k,n}(u)$$

with $p_{j,k}$ specifying the location of (m+1) by (n+1) control points

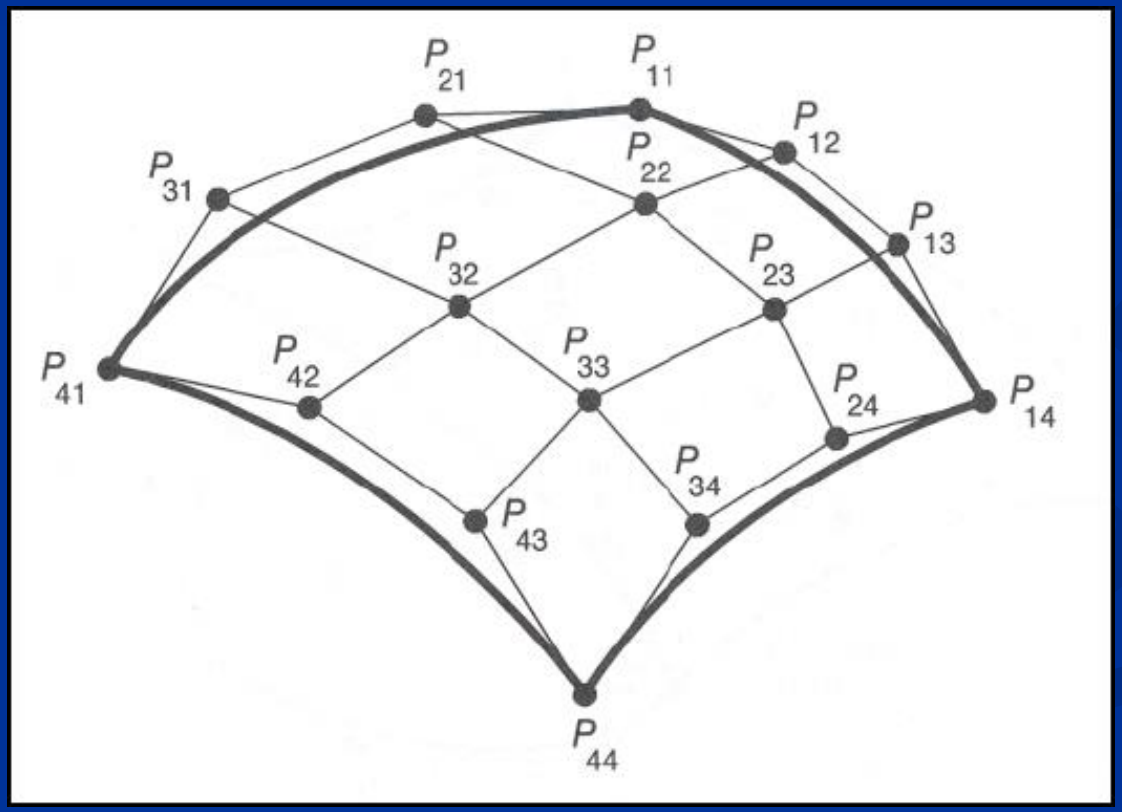
Bezier Surfaces

- u and v parameters



Bezier Surfaces

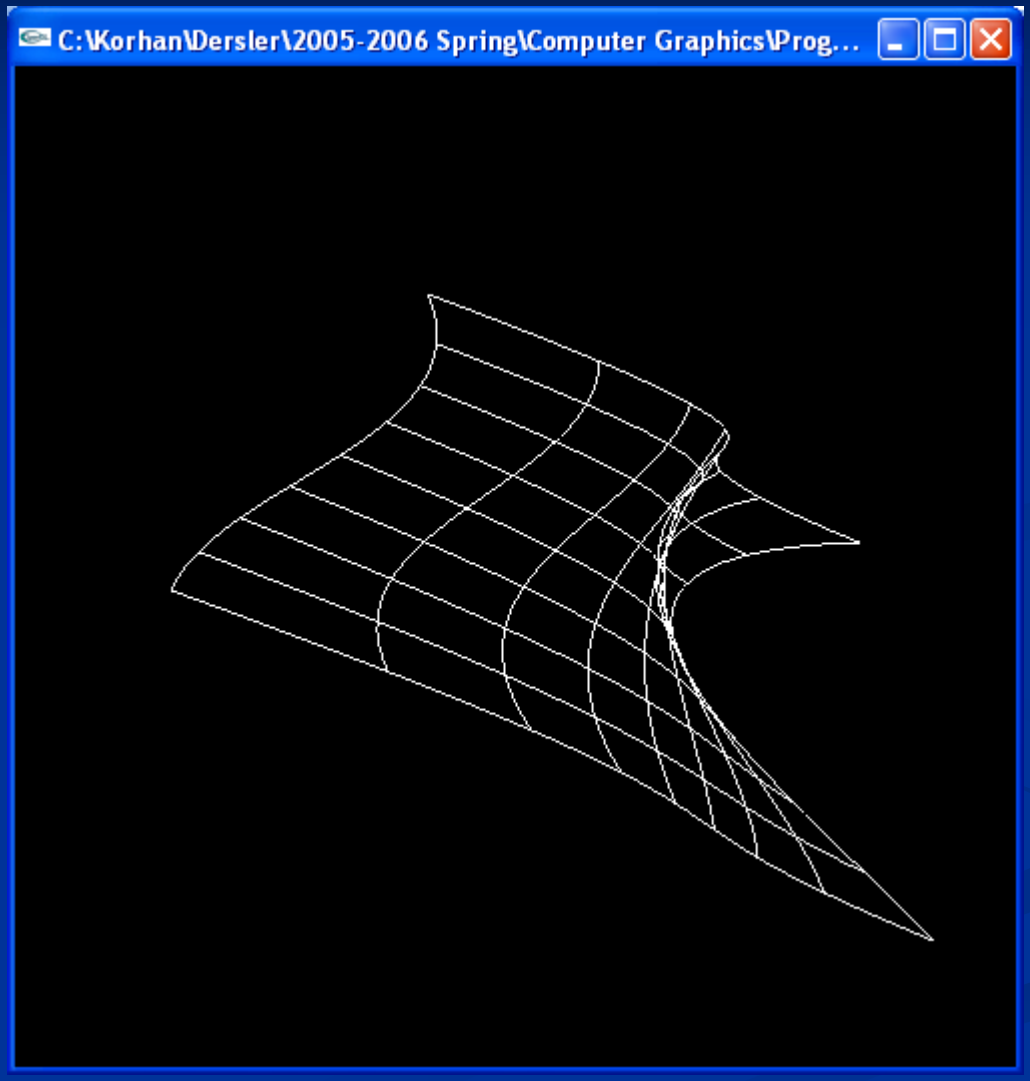
- An example Bezier surface



OpenGL Bezier-Spline Surface Functions

- We specify parameters and activate the routines for Bezier surface display with
  - glMap2*(GL_MAP2_VERTEX_3, uMin, uMax, uStride, nuPts, vMin, vMax, vStride, nvPts, *ctrlPts);
  - glEnable(GL_MAP2_VERTEX_3);
- and deactivate with
  - glDisable(GL_MAP2_VERTEX_3);
  - uMin, uMax, vMin and vMax are typically 0 and 1.0
  - stride=3 for 3D
  - nuPts and nvPts are the size of the array
  - ctrlPts is the double subscripted array of control points

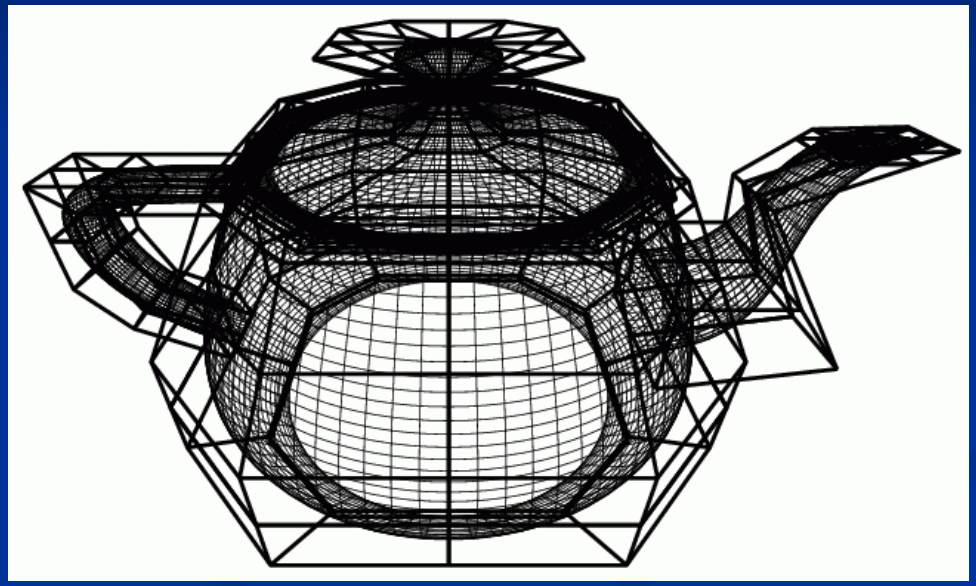
Example OpenGL Code

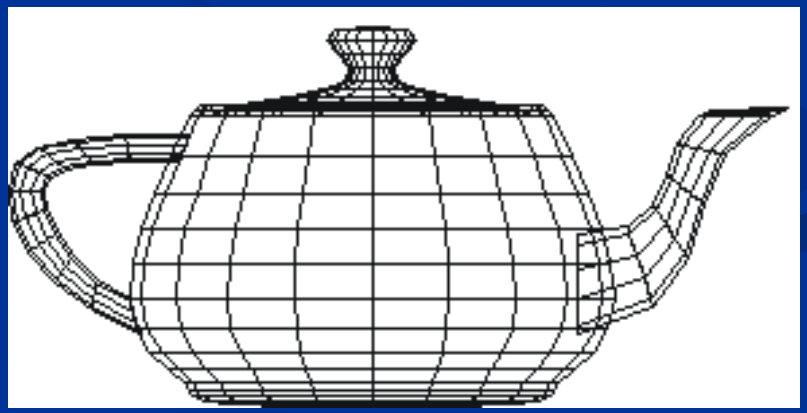


bezsurf.c

Bézier Surfaces: Example

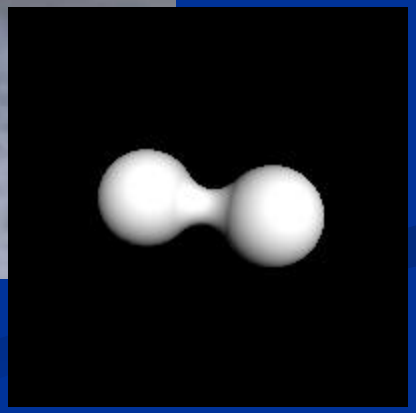
- Utah Teapot modeled by 32 Bézier Patches



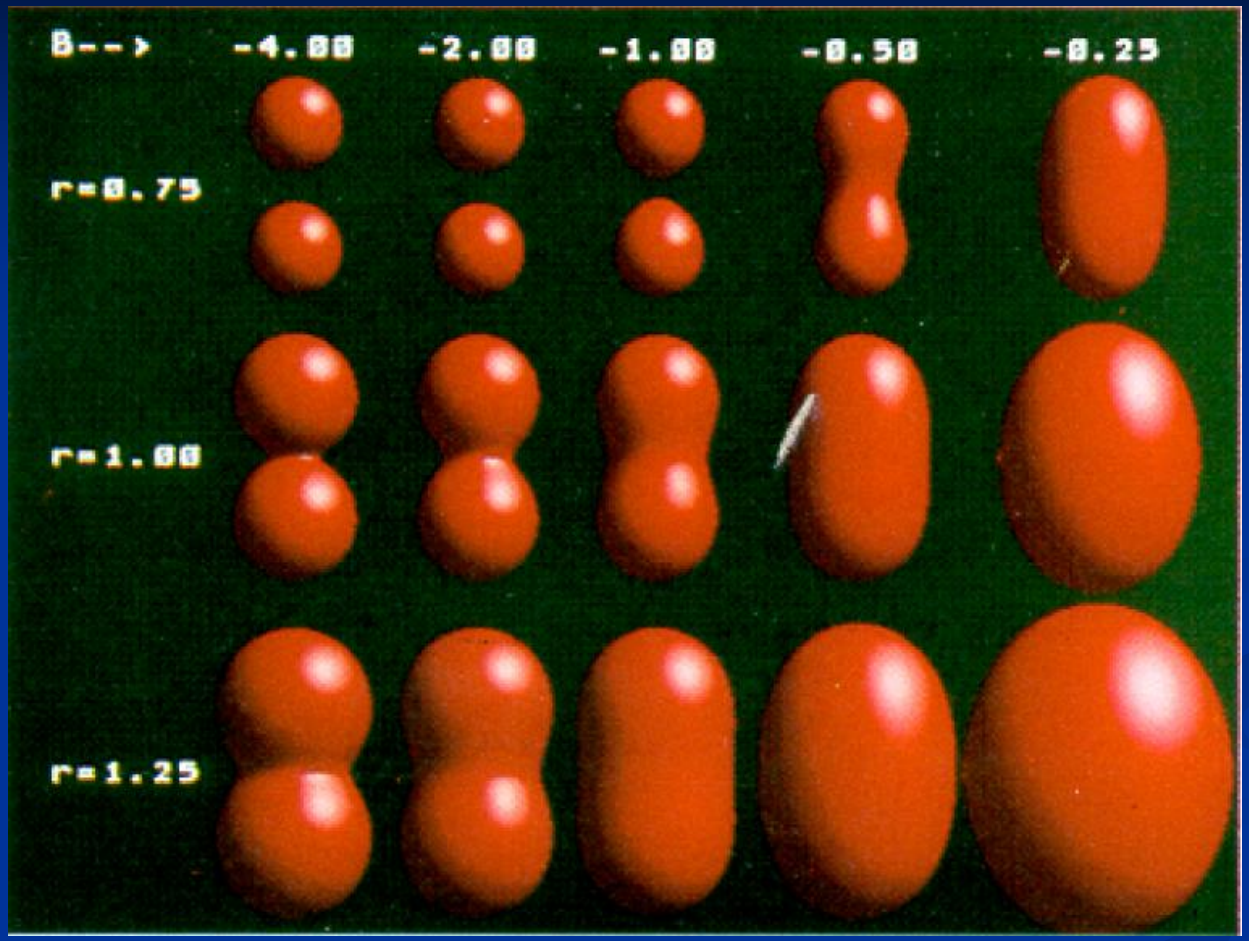


Blobby Objects

- Memodelkan objek yang dapat berubah bentuk tapi volumenya tetap
- Contoh
  - Water drops
  - Molecules
  - Force fields

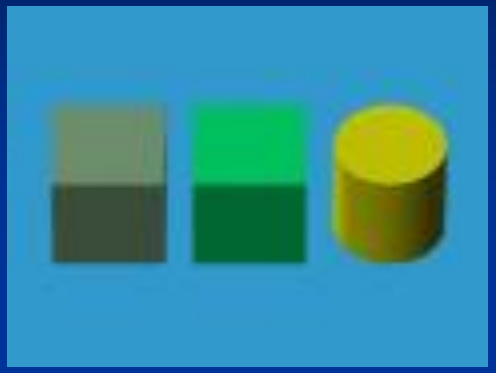


Metaballs (Blinn Blobbies)

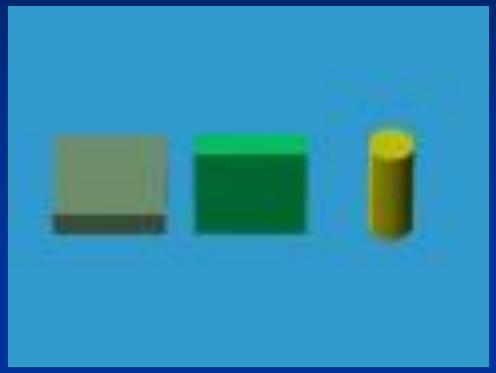


- Blobby objek dengan beberapa nilai radius r dan jarak B
- 2 obyek blobby sifatnya seperti molekul, semakin dekat jaraknya maka akan menyatu

Constructive Solid Geometry



Primitives



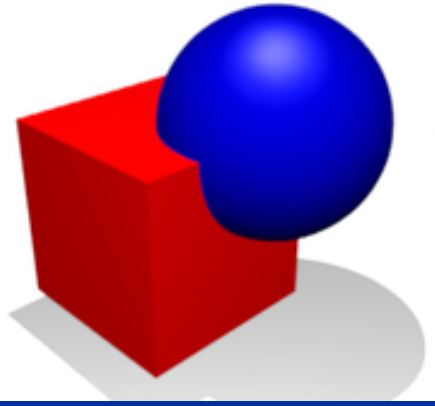
Transformed

Combined

Bermula dari objek geometri primitive, ditransformasikan dan dikombinasikan membentuk objek yang kompleks

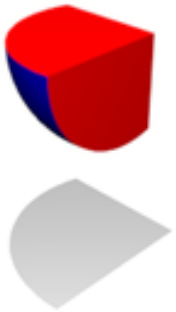
Operations in constructive solid geometry

Boolean union



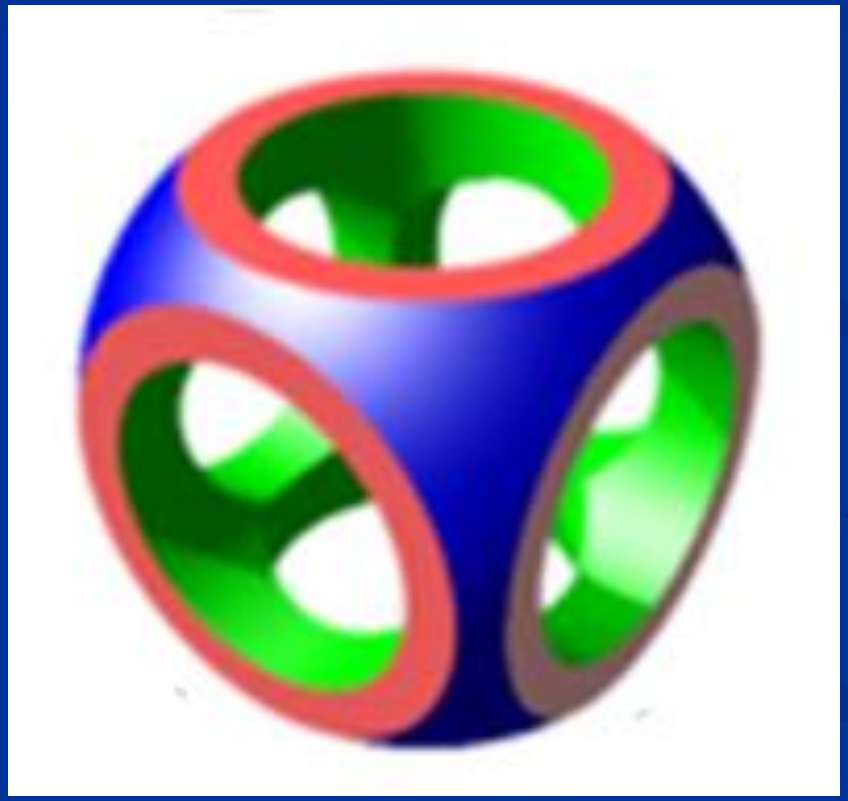
Boolean difference

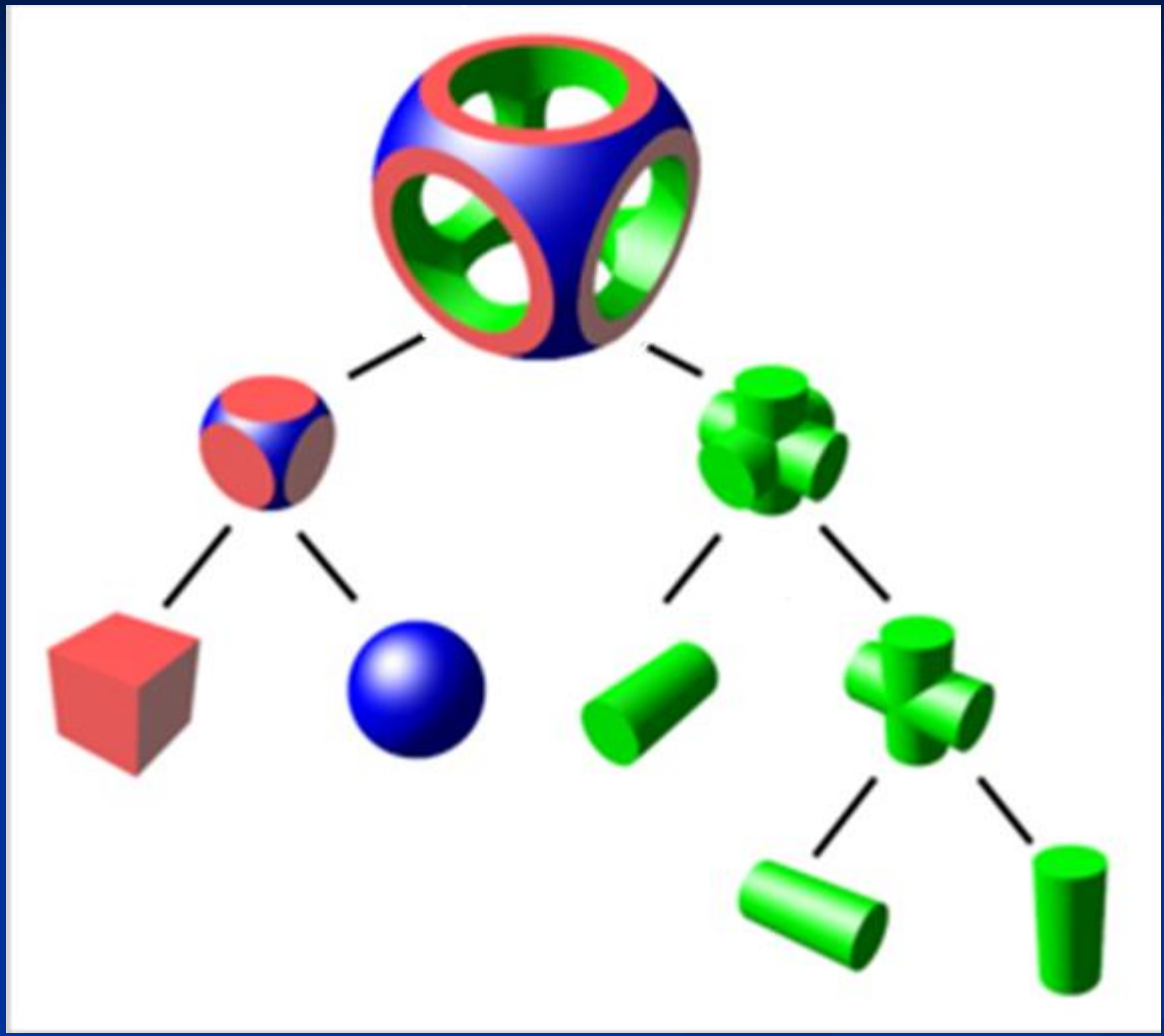
Boolean intersection

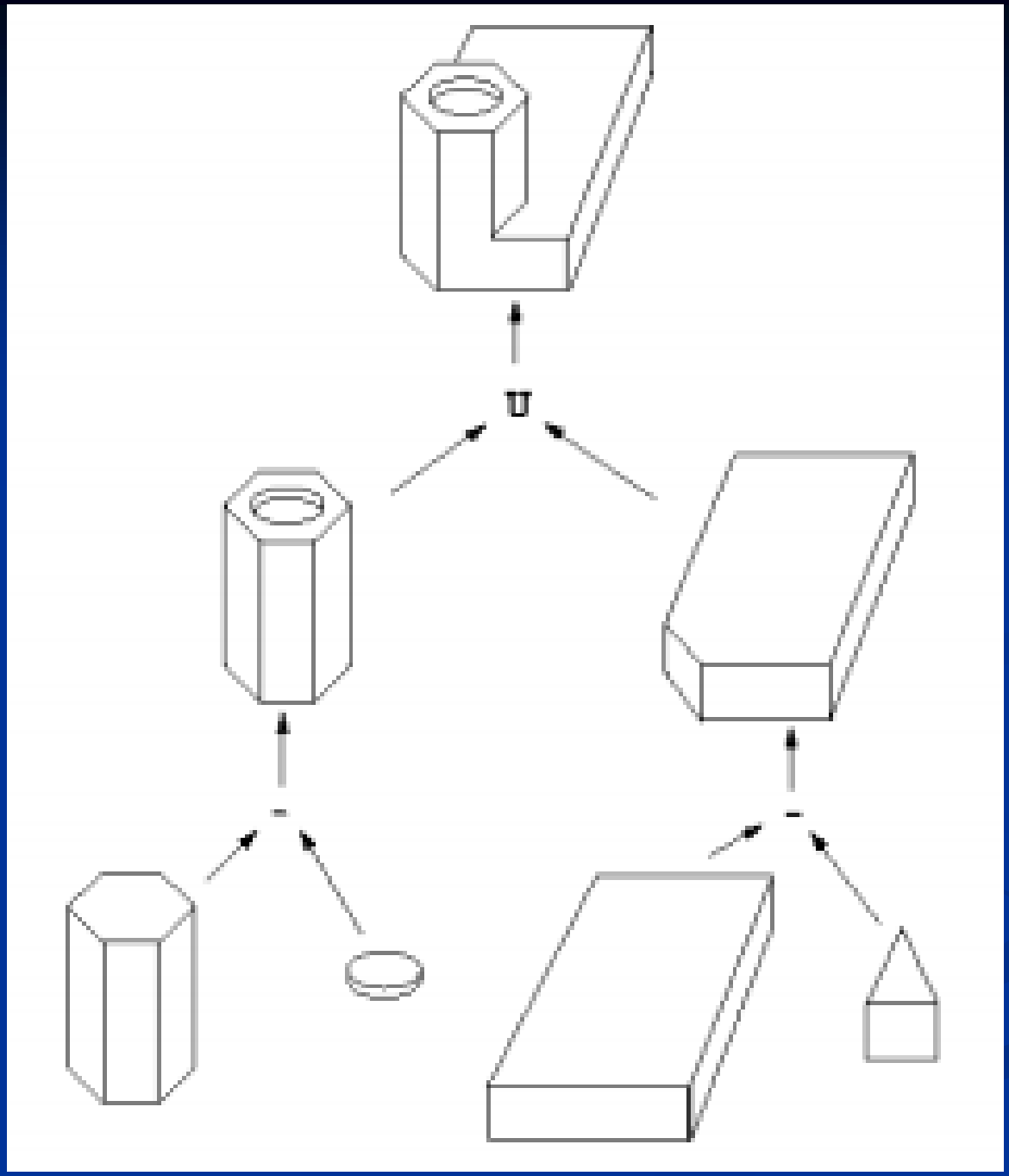


CGS

- Bagaimana caranya membuat objek berikut?

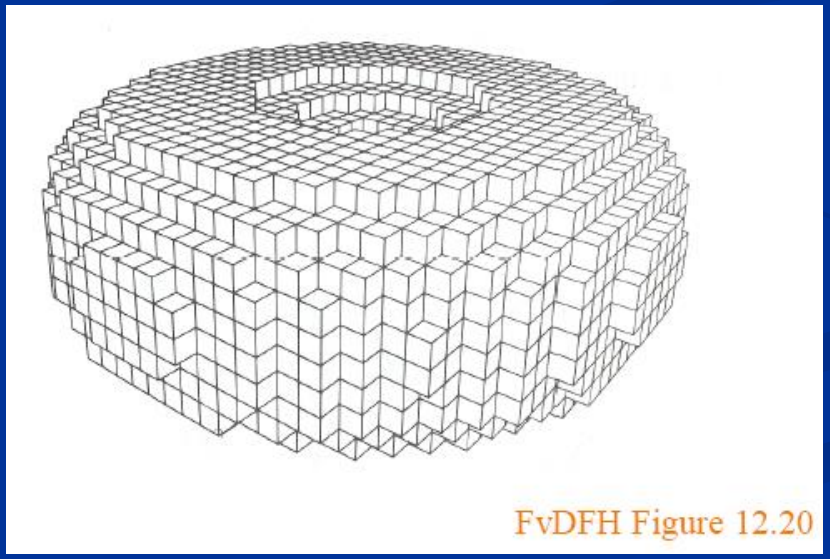






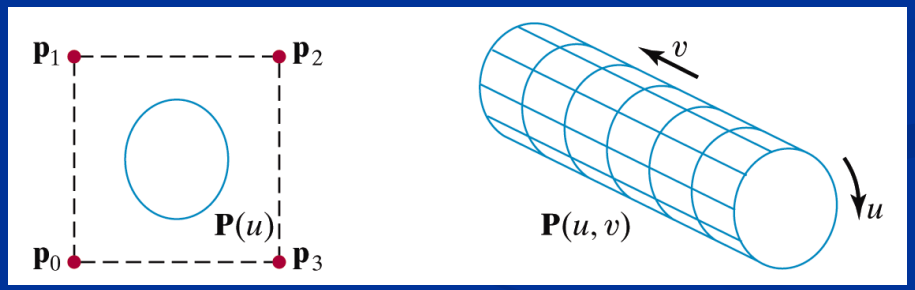
Voxel

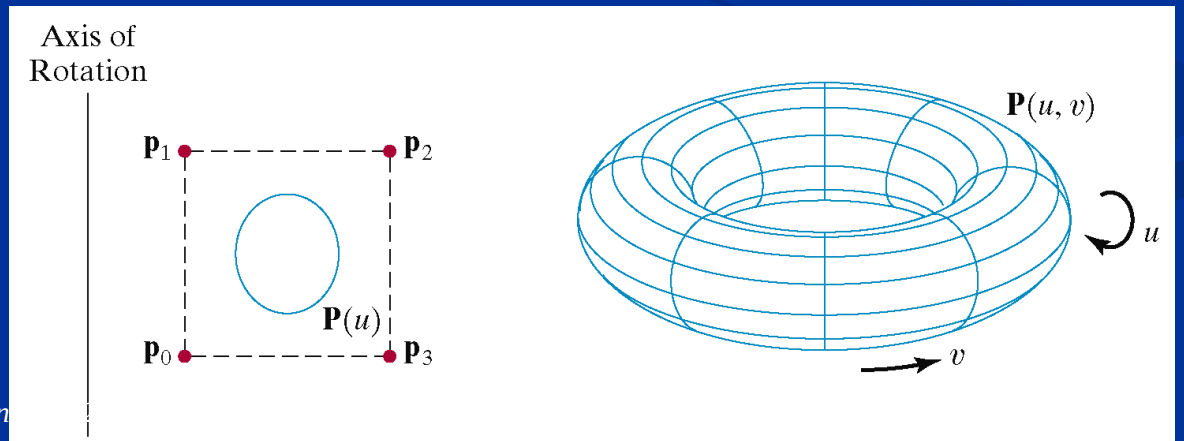
- Membagi ruang menjadi sekumpulan grid berukuran sama
  - Setiap sel pada grid disebut *voxel* (ingat istilah *pixel*)



Sweep (Extrusion)

- Membuat objek 3D dari bidang 2D yang di-sweep pada sumbu atau lintasan tertentu
- Dapat juga membentuk bidang 2D dari suatu kurva





Contoh Bezier Curve

- Diberikan suatu kurva bezier derajat 1 dengan titik awal (-2,4) dan titik akhir (-2,4) dan titik kontrolnya adalah (2,2).
- Buatlah plotting titik pada kurva sebanyak 6 titik (termasuk titik awal dan akhir).

- Terima kasih

Lighting

- Rendering
- Light source
- Reflection models
- Shading models

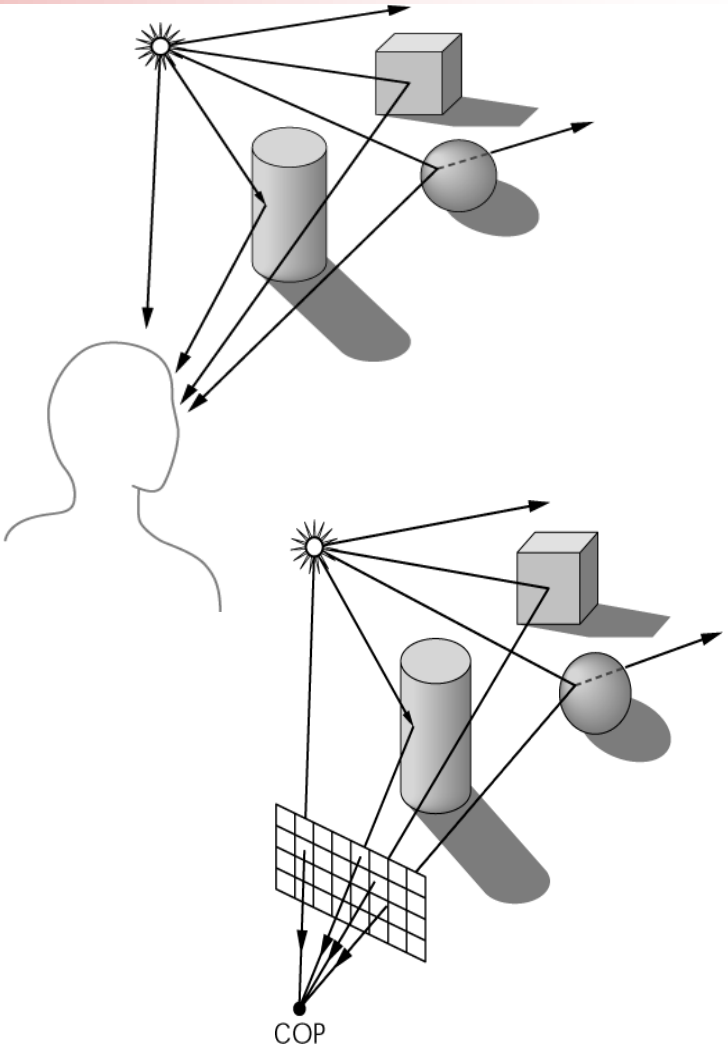
Rendering

- Concerned with determining the most appropriate colour (i.e. RGB tuple) to assign to a pixel associated with an object in a scene

- We need to know
  - how to describe light sources
  - how light interacts with materials - reflection models
  - how to calculate the intensity of light that we see at a given point on object surface - shading models

A Model for Lighting

- Only light that reaches the viewers eye is ever seen
- Direct light is seen as the colour of the light source
- Indirect light depends on interaction properties
- In computer graphics we replace viewer with projection plane
- Rays which reach COP after passing through viewing plane are actually seen



Illumination Variables

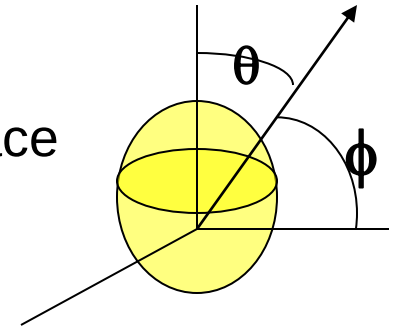
- Light source
  - Positions
  - Properties
- Object
  - Geometry of the object at that point (normal direction)
  - Material properties
    - opaque/ transparent, shiny/dull, texture surface patterns
- Position and orientation of view plane

Modelling the Light Source

- Light sources characterized by the illumination function:

$$I(x, y, z, \theta, \phi, \lambda)$$

(x, y, z) : point on light source surface
(θ, ϕ) : direction of emission
λ: wavelength



- Contribution of a light source can be determined by integrating over the surface of the light source

- For real-time speeds it is easier if we can approximate with point source (or set of point sources)

Light sources

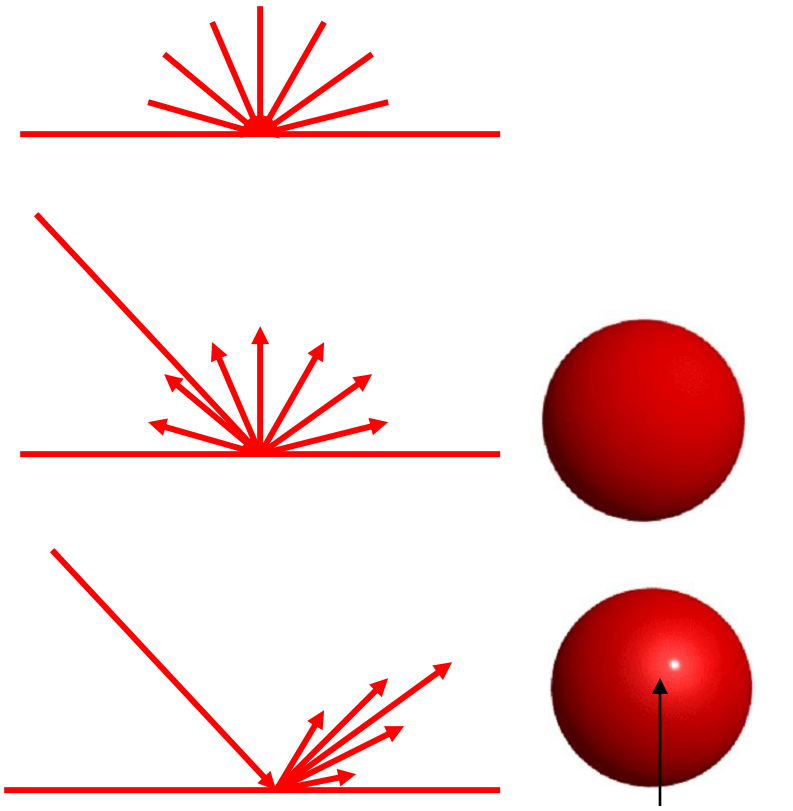
- Ambient light – uniform lighting
- Point source – emits light equally in all directions
- Spotlight – characterized by a narrow range of angles through which light is emitted
- Distance light sources – parallel rays of light

Reflection Model

- A reflection model (also called lighting or illumination model) describes the interaction between light and a surface
- The nature of interaction is determined by the material property
- Three general types of interaction: specular reflection, diffuse reflection, transmission

Reflection Model

- **Ambient** - reflected from other surfaces
- **Diffuse** - from a point source reflected equally in all directions
- **Specular** - from a point source reflected in a mirror-like fashion



The Phong Reflection Model

- The Phong Illumination Model is a local illumination model and is largely an empirical model. However it is fast to compute and gives reasonably realistic results.

- Light incident upon a surface may be reflected from a surface in two ways:

  - **Diffuse reflection**: Light incident on the surface is reflected equally in all directions and is attenuated by an amount dependent upon the physical properties of the surface. Since light is reflected equally in all directions the perceived illumination of the surface is not dependent on the position of the observer. Diffuse reflection models the light reflecting properties of matt surfaces.

  - **Specular Reflection**: Light is reflected mainly in the direction of the reflected ray and is attenuated by an amount dependent upon the physical properties of the surface. Since the light reflected from the surface is mainly in the direction of the reflected ray the position of the observer determines the perceived illumination of the surface. Specular reflection models the light reflecting properties of shiny or mirror-like surfaces.

The Phong Reflection Model

- A local illumination model including only contributions from **diffuse** and **specular** components suffers from one large drawback, namely, a surface that does not have light incident on it will reflect no light and will therefore appear black.

- This is not realistic, for example a sphere with a light source above it will have its lower half not illuminated. In practice in a real scene this lower half would be partially illuminated by light that had been reflected from other objects. This effect is approximated in a local illumination model by adding a term to approximate this general light which is `bouncing' around the scene. This term is called the **ambient reflection** term and is modelled by a constant term. Again the amount of ambient light reflected is dependent on the properties of the surface.

- Hence the local illumination model that is generally used is

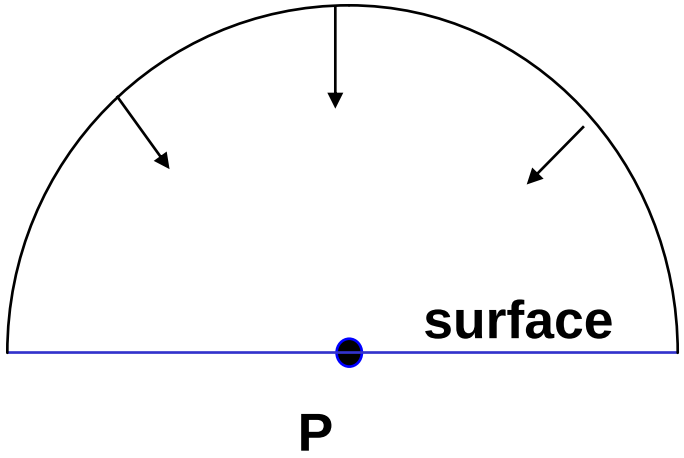
  illumination = Ambient + Diffuse + Specular

Ambient Reflection

In the Phong model ambient light is assumed to have a constant intensity throughout the scene. Each surface, depending on its physical properties, has a coefficient of ambient reflection which measures what fraction of this light is reflected from the surface. Hence for an individual surface the intensity of ambient light reflected is:

$$I = K_a L_a$$

I = Reflected intensity
K_a = Reflection coefficient
I_a = Ambient light intensity (same at every point)

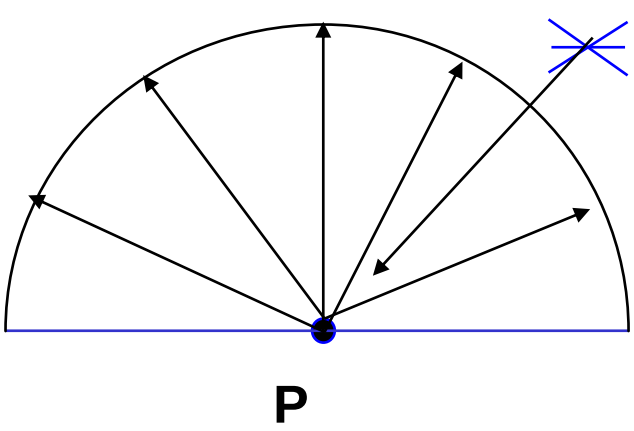


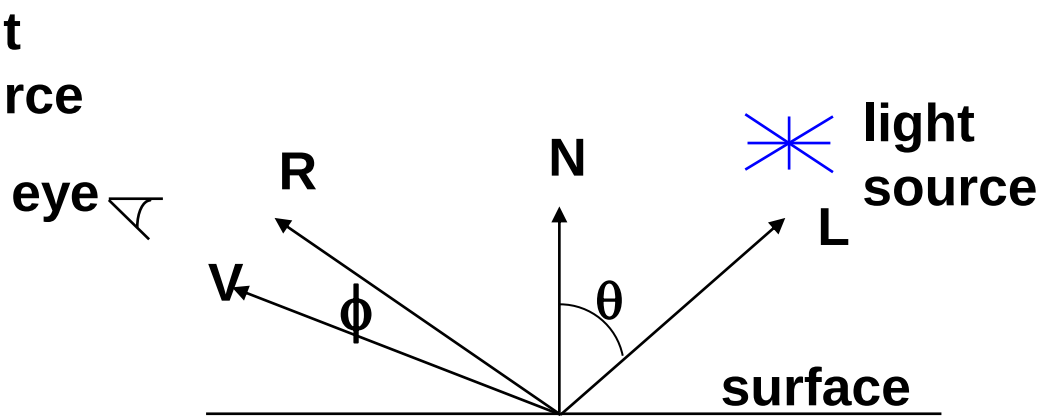
Diffuse Reflection

A perfectly diffuse reflecting surface scatters light equally in all directions. Thus the intensity at a point on a surface as perceived by the viewer does not depend on the position of the viewer.

The colour of the light reflected from the surface depends upon the colour of the light and the properties of the surface. Light incident on the surface will have some components absorbed and others scattered thus giving the surface its colour. Thus a surface that appears red under white light absorbs green and blue and scatters red light. When only diffuse light is considered surfaces will appear dull or matt.

Diffuse Reflection





The intensity due to diffuse reflection is given by Lambert's cosine law:

$$\mathbf{I_d = K_d\, I_i \cos\theta}$$

$\mathbf{I_d}$ = Reflected intensity
K_d = Diffuse reflection coefficient
I_i = Intensity of the incident light

If there is more than one light source then the diffuse intensity is summed over all light sources.

The Effect of Distance

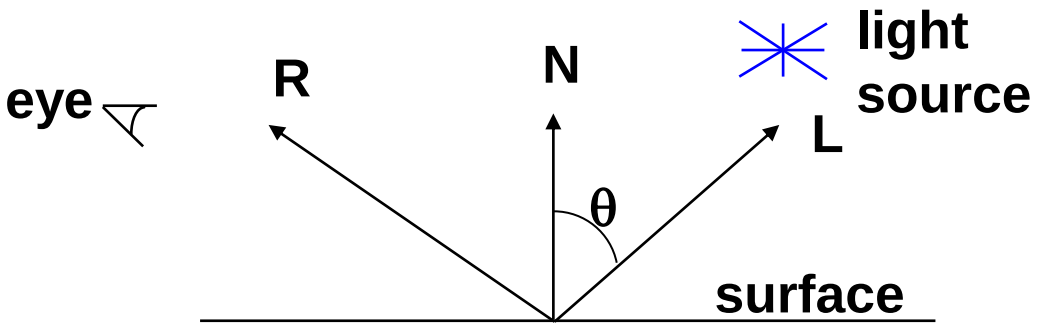
With the lighting model proposed so far two surfaces with the same properties and orientation but different distances from the light source would have the same intensity of illumination. This can be corrected by including a factor dependent on the *distance of the surface point to the viewing point*. Hence the lighting model (for a single light source) and modelling only ambient and diffuse reflection is now:

$$I = K_a I_a + \frac{K_d I_i \cos\theta}{r + k}$$

where r is the distance and k is an arbitrary constant chosen to make the image appear correct.

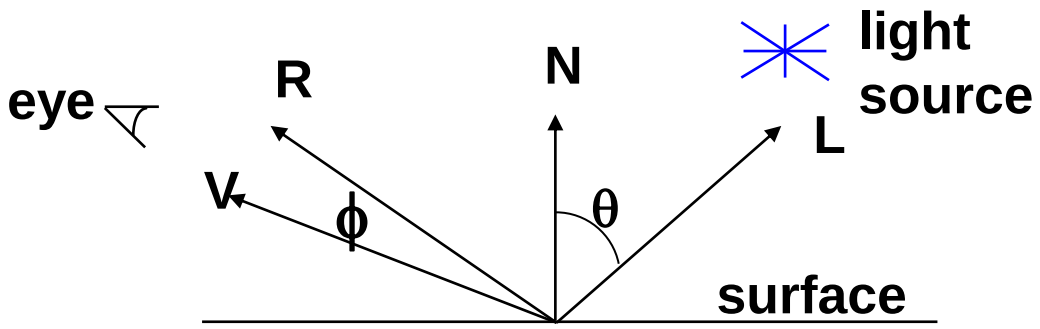
Specular Reflection

- Specular reflection is caused by the mirror-like properties of a surface. A perfect mirror will reflect light arriving at the surface at an angle of incidence *theta* to the normal at a reflected angle of *theta* to the normal in the same plane as the normal and the incident light. This means that only a viewer on the reflected ray will actually see the reflected light.



Specular Reflection

- In practice no surface is a perfect mirror and there will be a certain amount of light scattered around the reflected direction.

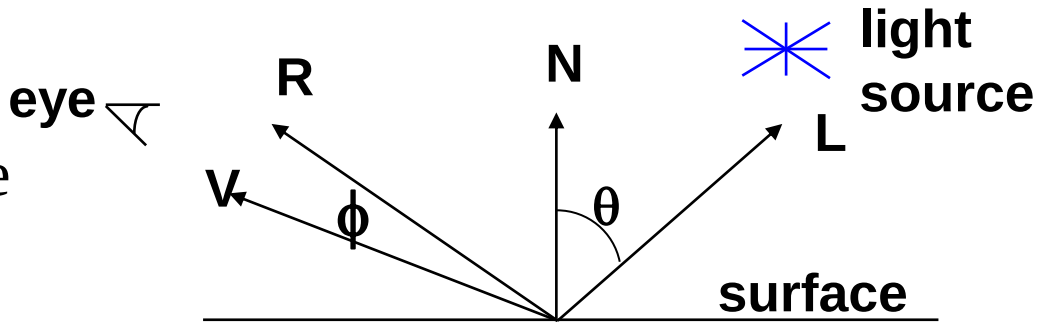


- The reflected light is therefore seen over an area of the surface as a **highlight**. The colour of this specularly reflected highlight is usually taken to be that of the light source. In diffuse reflection the reflected light is the colour of the surface.

- In practice the distribution function for specularly reflected light is a complex function of *Phi* - the angle between the reflected ray and the viewing direction **V**.

Specular Reflection

- Amount of light visible to viewer depends on the angle ϕ between R and V:



- $I_s = K_s I_i (\cos\phi)^n$

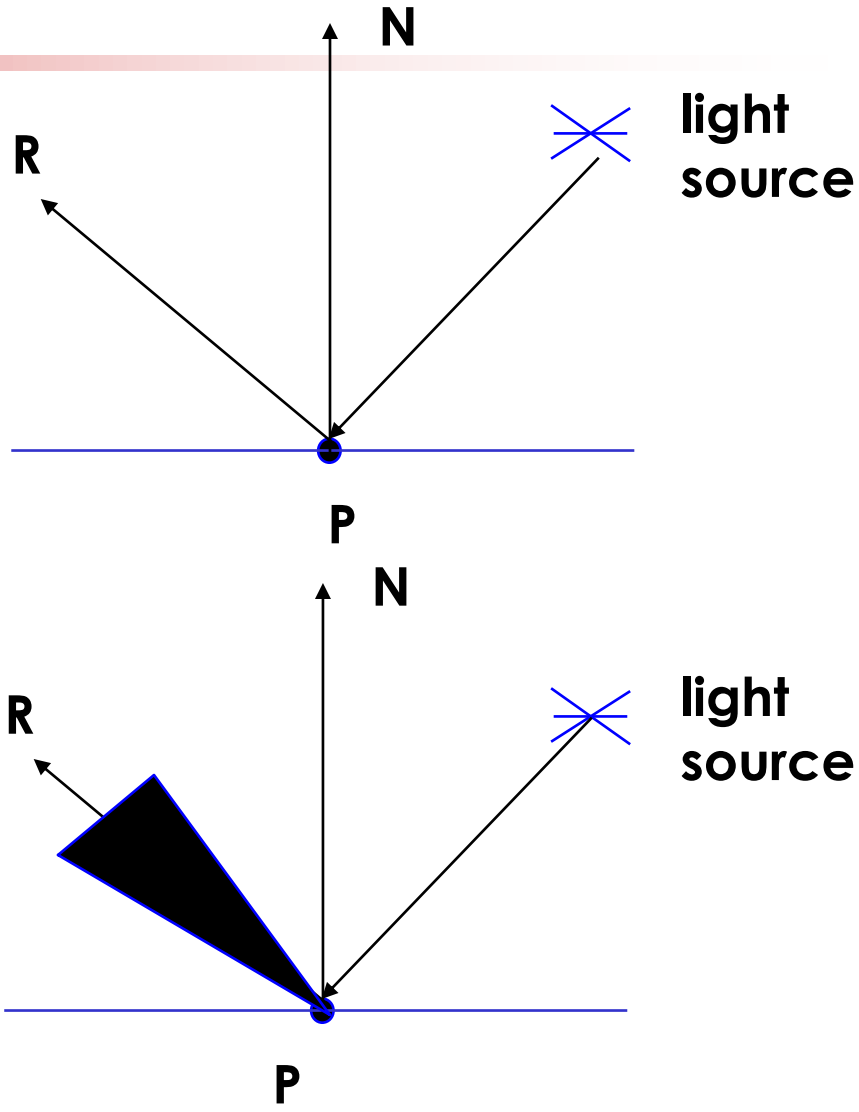
- n varies with material, large n: shiny, small n: dull

V = Direction to the viewer (COP)
R = Direction of perfect reflected light
N = Surface normal
L = Direction of light source
I_i = Intensity of the incident light
K_s = Specular reflection coefficient
$\mathbf{I_s}$ = Reflected intensity

Specular Reflection

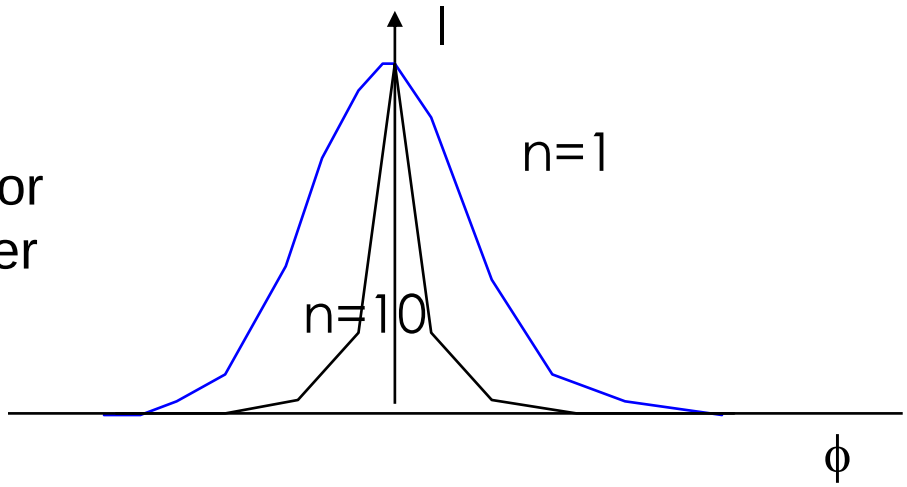
n perfect specular reflection, light
s reflected along the direction
symmetric to the incoming light

n practice, light is reflected within
a small angle of the perfect
eflection direction - the intensity
of the reflection tails off at the
outside of the cone. This gives a
narrow highlight for shiny
surfaces, and a broad highlight
or dull surfaces



Specular Reflection

- Thus we want to model intensity, I, as a function of angle between viewing direction and the reflection, say ϕ, with a sharper peak for shinier surfaces, and broader peak for dull surfaces



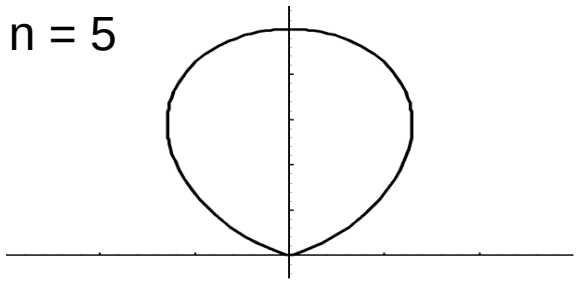
- This effect can be modelled by $(\cos\phi)^n$, with a sharper peak for larger n

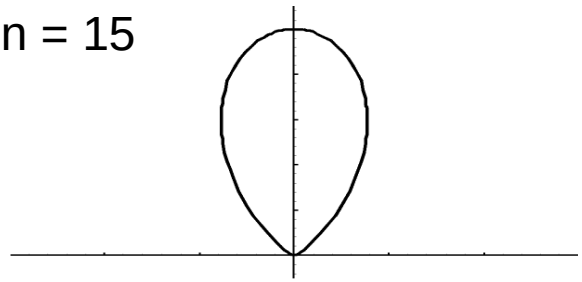
- Empirical

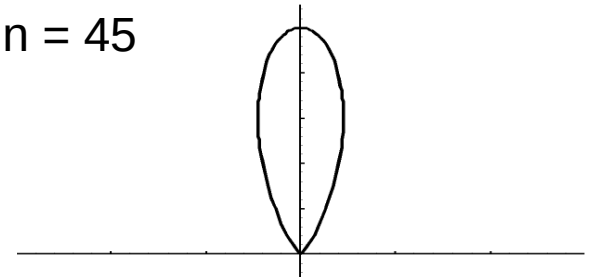
Specular Highlights

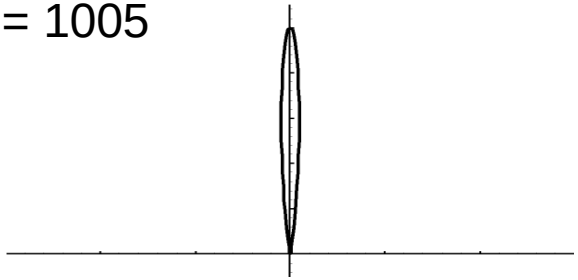
- The cosine function (defined on the sphere) gives us a lobe shape which approximates the distribution of energy about a reflected direction controlled by the shinyness parameter a known as the Phong exponent.

$(\cos\phi)^n$ ⟶

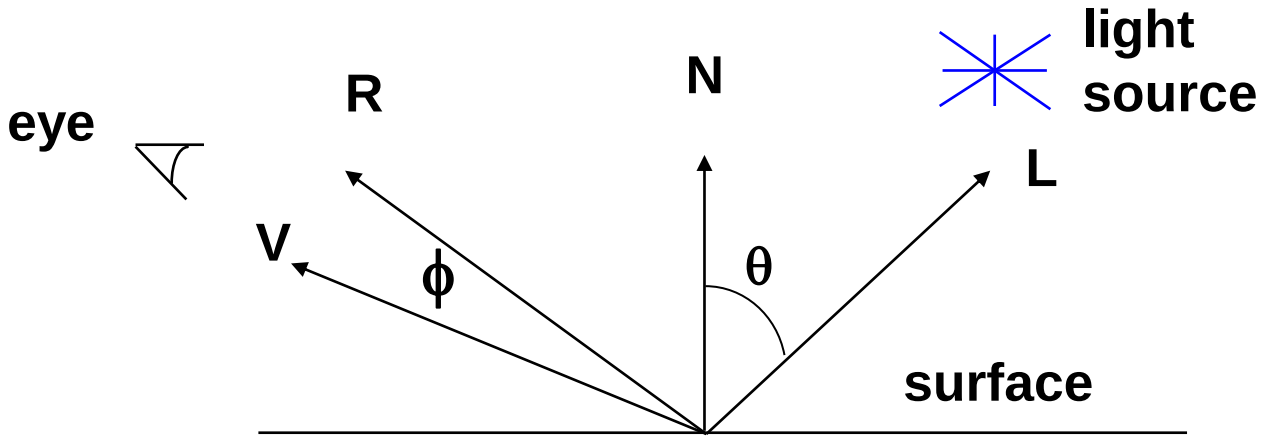








Ambient, Diffuse and Specular



Hence we obtain the complete basic reflection model:

$$I = K_a I_a + \frac{K_d I_i (L \bullet N) + K_s I_i (R \bullet V)^n}{r + k}$$

Note that it is assumed that N, R and V are unit vectors

The coefficient of Specular Reflection is treated as a constant in the Phong model. However it is actually a function of the angle of incidence.

Colour

- To handle colour the usual approach is to treat specular highlights as being the same colour as the light source.

- The colour of objects is handled by treating the coefficients of ambient and diffuse reflection as having red, green and blue components. Hence the red, green and blue signals that drive the display are produced from:

$$I_r = K_{ar} I_a + \frac{K_{dr} I_i (L \bullet N) + K_s I_i (R \bullet V)^n}{r + k}$$

$$I_g = K_{ag} I_a + \frac{K_{dg} I_i (L \bullet N) + K_s I_i (R \bullet V)^n}{r + k}$$

$$I_b = K_{ab} I_a + \frac{K_{db} I_i (L \bullet N) + K_s I_i (R \bullet V)^n}{r + k}$$

The Diffuse and Specular terms are summed over each light source

Summary of the Phong Model

- Light sources are usually considered as point sources situated infinitely far away. Hence the angle between the incident light and the normal to a planar surface is constant over a planar surface.
- The viewer is assumed positioned at infinity, hence the angle between the viewing direction and the reflected ray is constant over a planar surface.
- The diffuse and specular terms are modelled as local components only.
- An empirical result is used to model the distribution of the specular term around the reflection vector.
- The colour of the specular term is assumed to be that of the light source.
- The global illumination is modelled as a constant ambient term.
- Shadows are not handled.
- The coefficient of diffuse reflection is wavelength dependent. But it is sampled at each of the three primary colours.
- The coefficient of specular reflection is modelled as a constant but is actually dependent on the angle of incidence.

Local Shading Models

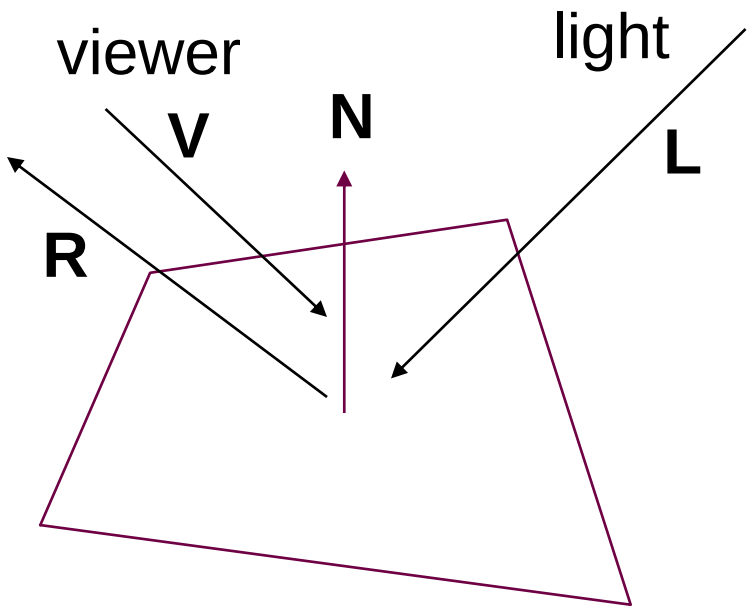
- Local shading models provide a way to determine the intensity and color of a point on a surface
- They do not require knowledge of the entire scene, only the current piece of surface.
- For the moment, assume:
  - We are applying these computations at a particular point on a surface
  - We have a normal vector for that point

Flat Shading

- Compute shading at a representative point and then generalise to the whole polygon
  - For a flat polygon the vector n is constant across the surface
  - If we assume a distant viewer, v is constant
  - If we assume a distant light source l is constant
- If the three vectors are constant then the shading calculation needs to be calculated only once for each polygon

Flat Shading

- Calculate normal
- Assume L.N and R.V constant (light & viewer at infinity)
- Calculate I_r, I_g, I_b using Phong reflection model
- Project vertices to viewplane
- Use scan line conversion to fill polygon

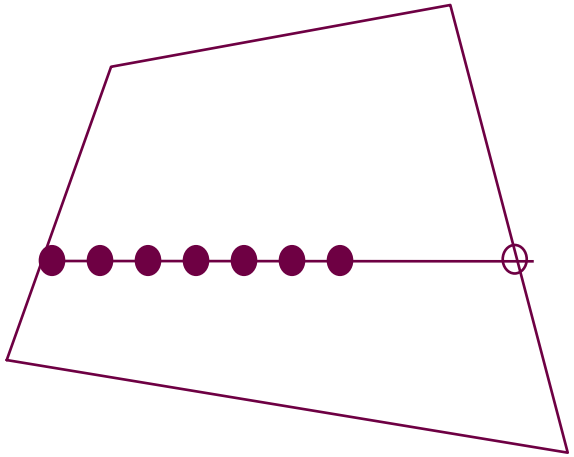


Flat Shading

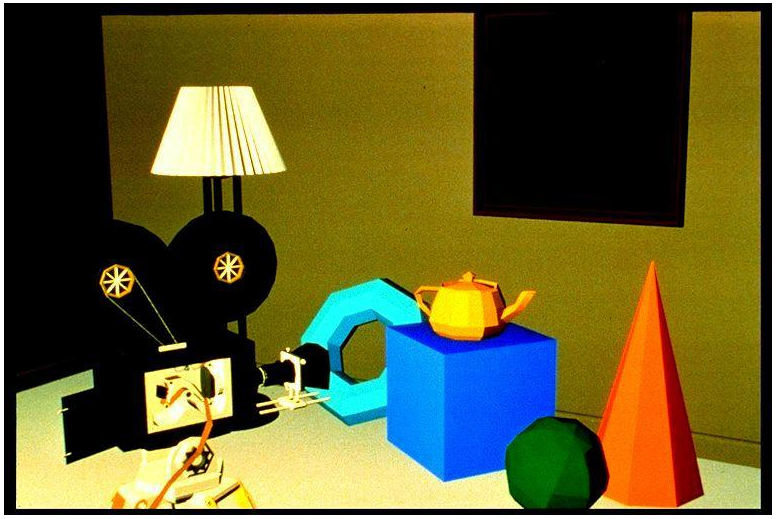
- When flat shading is used the same shade is used over the whole surface. If the surface actually corresponds to a planar surface in the model then this is correct. Though remember that as the Phong Illumination Model assumes that the light source is at infinity a large surface illuminated by a light positioned close to the surface will not be evenly illuminated in reality.
- The shade to be associated with a surface is calculated before View Transformation and Perspective Transformation are carried out. That is before all the angles are distorted.
- If flat shading of a planar surface is to be considered then an arbitrary point on the surface can be chosen at which to calculate the shade.
- If a curved surface is approximated by a mesh of polygonal facets then two approaches to calculating the shade of the facets can be used:
- Calculate the shade at some point on each facet, for example its centre.
- Calculate the shade at the vertices of the mesh using true surface normals. The shade at each facet can then be evaluated as the average of the shades at its vertices.
- Flat shading provides realistic images of objects which are composed entirely of planar surfaces but produce a facetted appearance when used to approximate curved surfaces.

2D Graphics - Filling a Polygon

- Scan line methods used to fill 2D polygons with a constant colour
  - find ymin, ymax of vertices
  - from ymin to ymax do:
    - find intersection with polygon edges
    - fill in pixels between intersections using specified colour



Flat Shading



Smooth Shading

- In many cases the polygons that are to be rendered in the polygon pipeline are an approximation to a curved surface.

- In this case the different shade calculated on each planar polygon will give a facetted appearance to the surface and it will not look like a smooth curved surface at all.

- To avoid this problem **incremental, or smooth, shading methods** are used to produce smoothly shaded pictures from planar polygon facetted representations of curved surfaces.

Smooth Shading

- The first step is to produce a shade value at each vertex of the polygonal mesh representing the surface. This can be done in two ways:

  1. Evaluate a surface normal on each polygonal facet. Produce a surface normal at each vertex by averaging the surface normals for the surrounding facets. The shade at the vertex can now be calculated.
  2. Evaluate the shade directly at each vertex by considering the vertex as a point on the actual curved surface and evaluating the actual surface normal to the surface at that point.

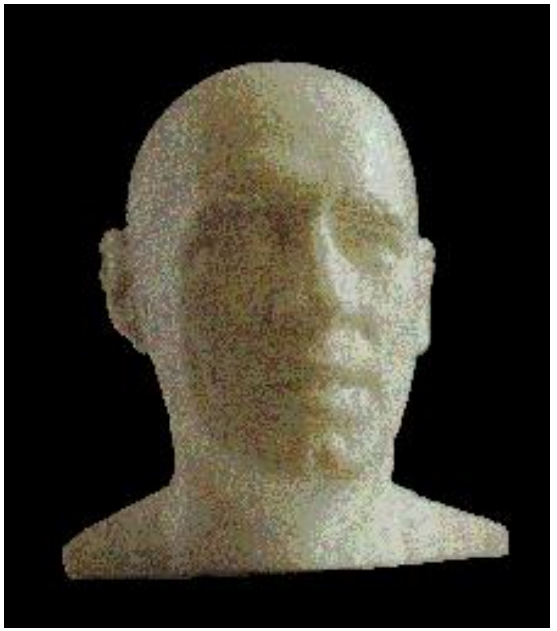
- Once the shade at the vertices of the polygonal mesh are known the shade at points interior to the polygonal facets are interpolated from the values at the vertices.

- This technique makes curved surfaces look `smooth shaded' even though based on a planar facet representation. The interpolation of shade values is incorporated into the polygon scan conversion routine.

- Hence an increase in realism is obtained at far less expense in computation than carrying out a pixel-by-pixel shading calculation over the whole original surface.

Gouraud Shading

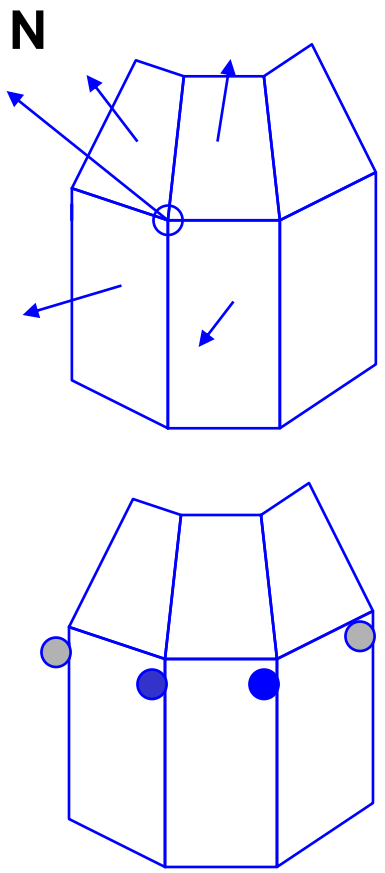
- Gouraud shading is a method for linearly interpolating a colour or shade across a polygon. It was invented by Gouraud in 1971
- It is a very simple and effective method of adding a curved feel to a polygon that would otherwise appear flat
- It can also be used to depth queue a scene, giving the appearance of objects in the distance becoming obscured by mist



Henri Gouraud is another pioneering figure in computer graphics

Gouraud Shading

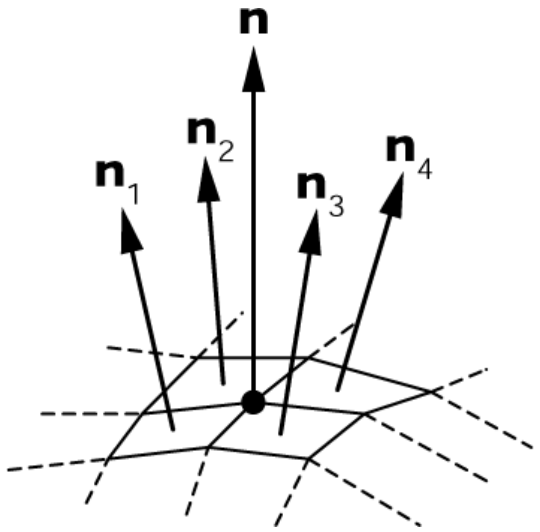
- Gouraud shading attempts to smooth out the shading across the polygons
- Unlike a flat shaded polygon, it specifies a different shade for each vertex of a polygon
- The rendering engine then smoothly interpolates the shade across the surface



Gouraud Shading

- Begin by calculating the normal at each vertex
- A feasible way to do this is by averaging the normals from surrounding polygons
- Then apply the reflection model to calculate intensities at each vertex

$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{\left|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4\right|}$$



Gouraud Shading

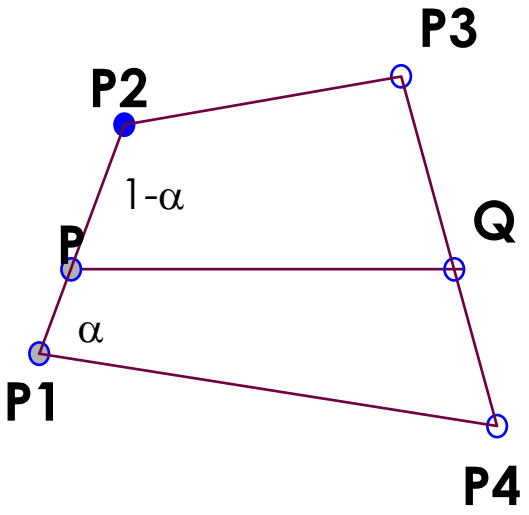
- We use linear interpolation to calculate intensity at edge intersection P

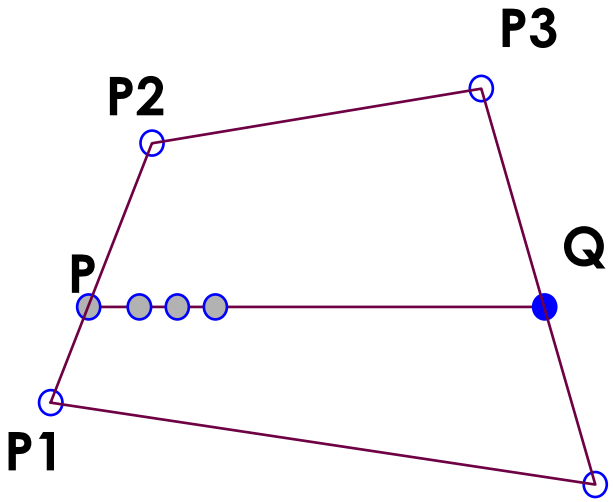
$$I_P^{RED} = (1-\alpha)I_{P1}^{RED} + \alpha I_{P2}^{RED}$$

where P divides P1P2 in the ratio α : 1-α

- Similarly for Q

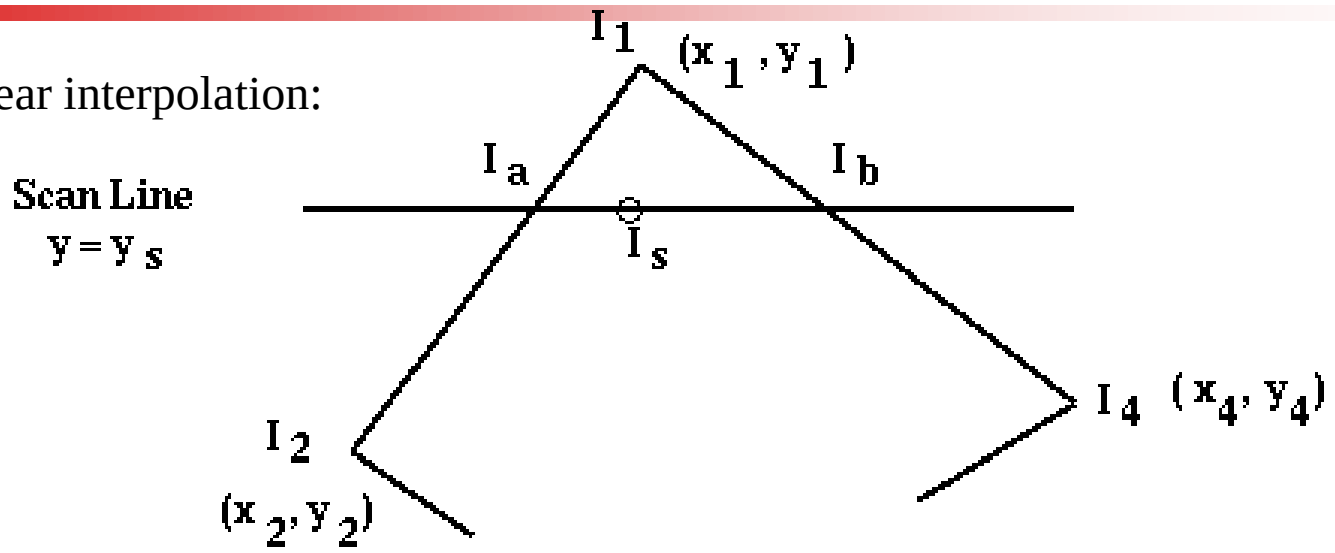
- Then we do further linear interpolation to calculate colour of pixels on scanline PQ





Gouraud Shading

- Bilinear interpolation:



$$I_a = \frac{1}{y_1 - y_2}(I_1(y_s - y_2) + I_2(y_1 - y_s))$$

$$I_b = \frac{1}{y_1 - y_4}(I_1(y_s - y_4) + I_4(y_1 - y_s))$$

$$I_s = \frac{1}{x_b - x_a}(I_a(x_b - x_s) + I_b(x_s - x_a))$$

Gouraud Shading Limitations

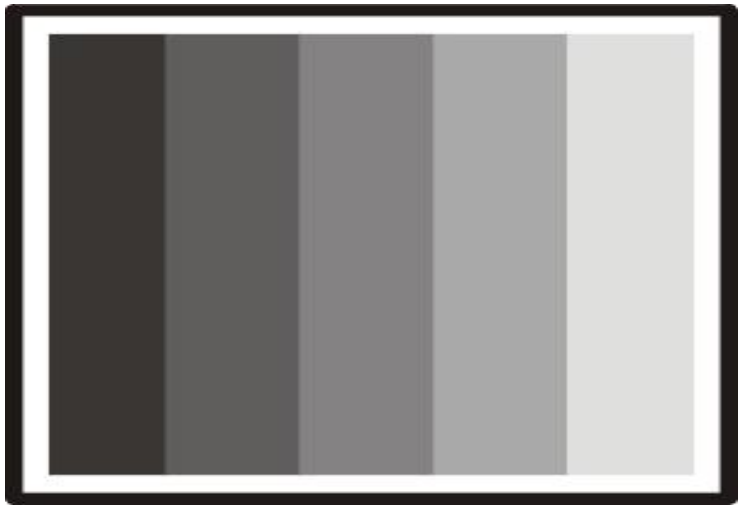
- Constant shading is simple, but it emphasizes the polygonal nature of the model. Our perceptual system exaggerates the effect because we see very clearly the boundary between adjacent light and dark shades (Mach banding)

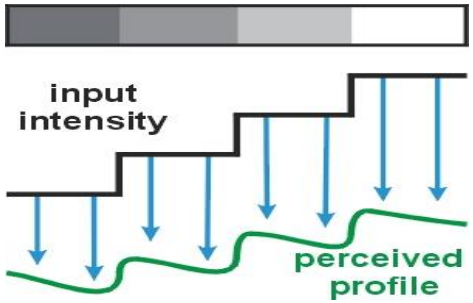
- Problems with Gouraud shading occur when you work with large polygons. A specular highlight at a vertex tends to be smoothed out over a larger area than it should cover - imagine you have a large polygon, lit by a light near it's center. The light intensity at each vertex will be quite low, because they are far from the light. The polygon will be rendered quite dark, but this is wrong, because it's centre should be brightly lit

- Gouraud shading can make a real difference over flat shaded polygons

Gouraud Limitations - Mach Bands

- The rate of change of pixel intensity is even across any polygon, but changes as boundaries are crossed
- This ‘discontinuity’ is accentuated by the human visual system, so that we see either light or dark lines at the polygon edges - known as Mach banding
- Mach bands where the intensity is constant in bands – similar effect with Gouraud shading





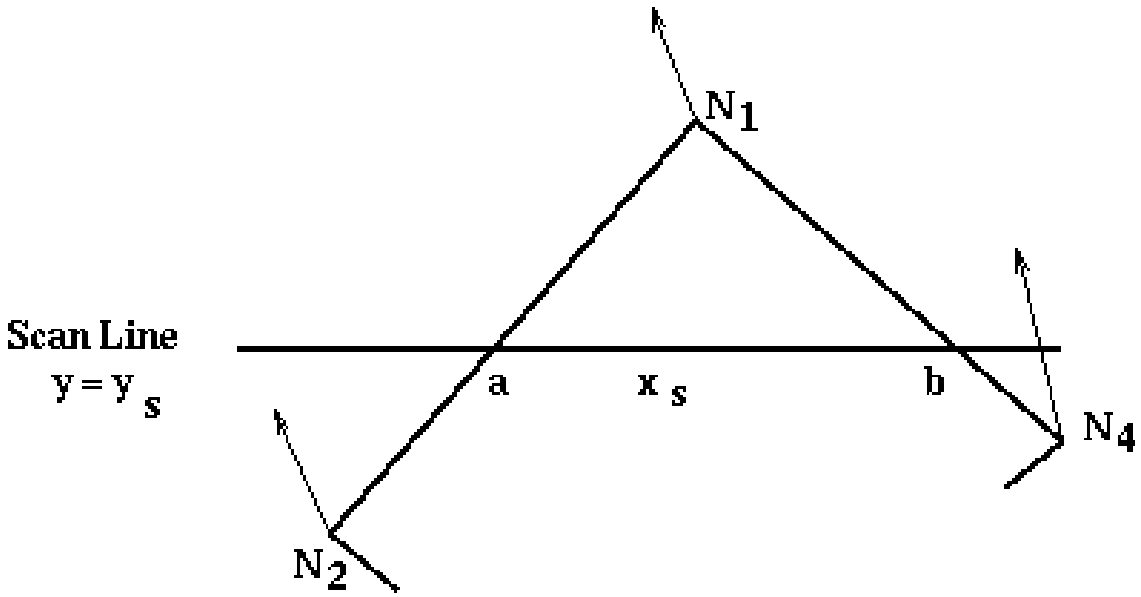
Phong Shading

- Phong shading uses similar principles to Gouraud shading but instead of interpolating the shade across the surface it is the **surface normals that are interpolated**. The interpolated normals are then used to evaluate a shade at each pixel.

- As before the light source and the viewer are assumed to be at infinity so that the intensity at a point is a function only of the interpolated normal.

- The equations for the interpolated normals are similar to Gouraud shading but each component of the normal has to be interpolated and the vector re-normalised after each interpolation step.

Phong Shading



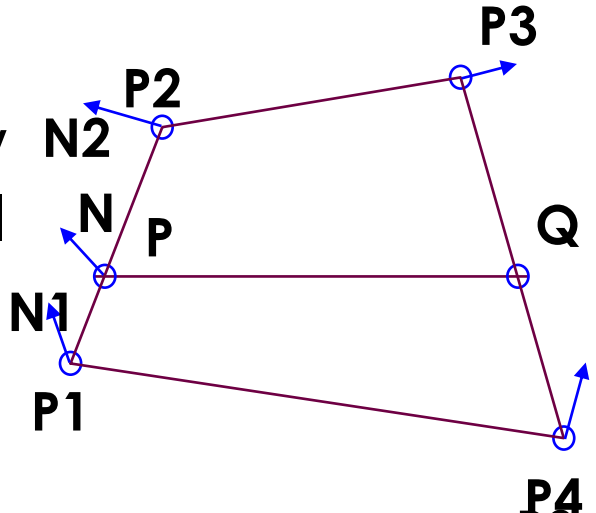
$$N_a = \frac{1}{y_1 - y_2}(N_1(y_s - y_2) + N_2(y_1 - y_s))$$

$$N_b = \frac{1}{y_1 - y_4}(N_1(y_s - y_4) + N_4(y_1 - y_s))$$

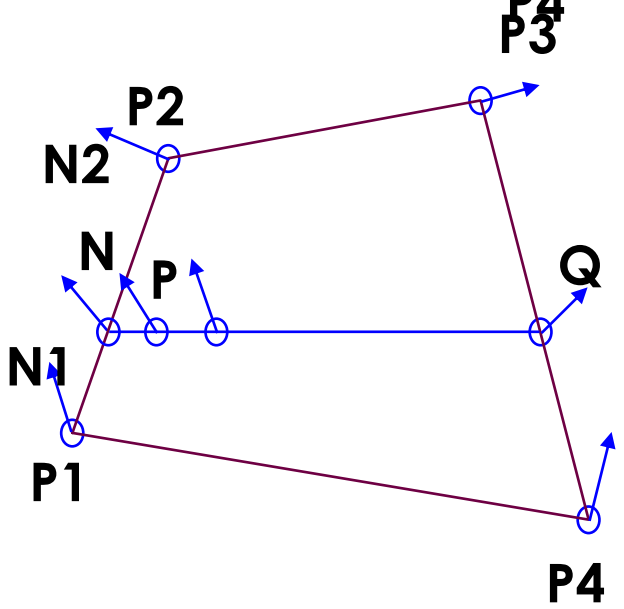
$$N_s = \frac{1}{x_b - x_a}(N_a(x_b - x_s) + N_b(x_s - x_a))$$

Phong Shading

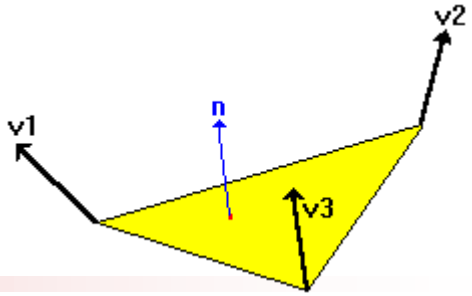
- Phong shading interpolates normals at each pixel, then apply the reflection model at each pixel to calculate the intensity IRED, IGREEN, IBLUE (shade each pixel individually)



- Every single pixel has it's brightness calculated using the interpolated normal vector

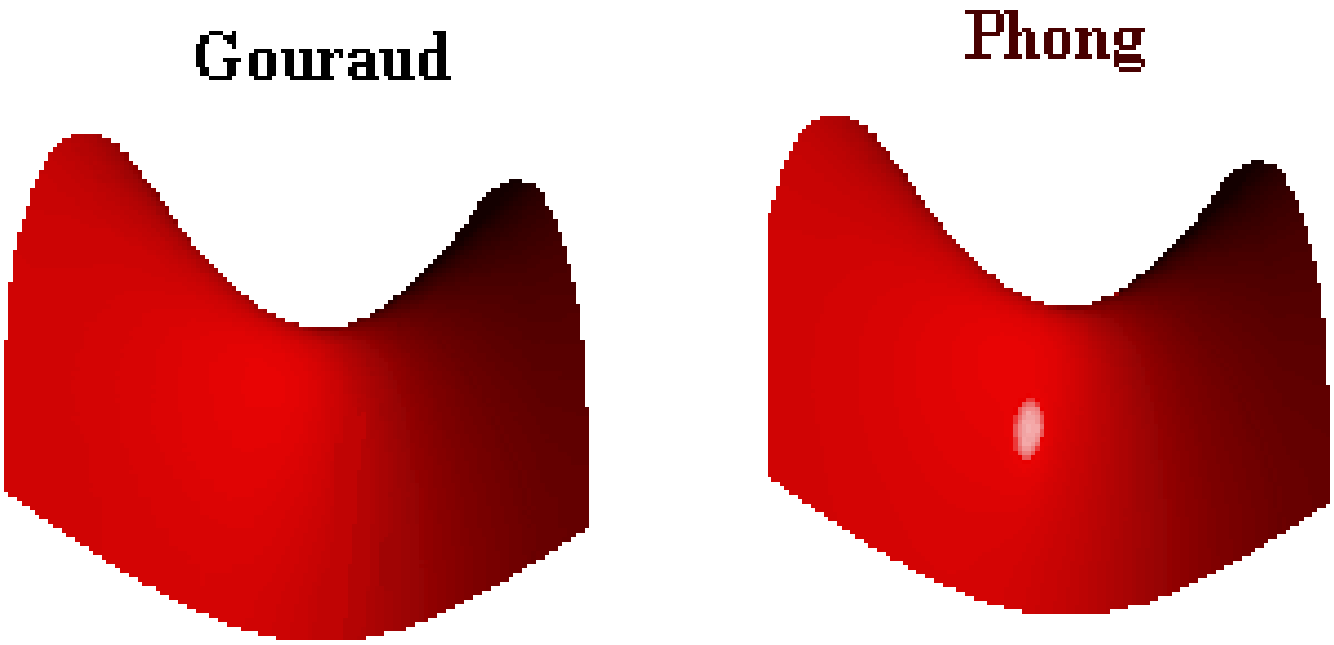


Phong Shading



- For a pixel on this polygon, the fraction of light from the light source reflecting off this pixel is the dot product of n and I.
- The dot product function returns values in the range 1 to -1. 1 is full brightness.
- There is no such thing as negative light, so values less then 0 should be taken as 0.
- If you multiply this value by the brightness of the light, then you have the brightness of that pixel

Gouraud versus Phong



Phong versus Gouraud Shading

- A major advantage of Phong shading over Gouraud is that specular highlights tend to be much more accurate, vertex highlight is much sharper
- The cost is a substantial increase in processing time because reflection model applied per pixel
- But there are limitations to both Gouraud and Phong
- In practice, some simplifications are made to the model for efficiency. For example, ambient light is sometimes assumed to be a constant

Flat, Gouraud and Phong Shading

Shading Summary

- It is expensive to calculate the intensity at each pixel of a projected polygon, and so there is a family of methods which use interpolation to calculate intermediate values
  - constant shading: reflection calculation is carried out once for each polygon, and the polygon is assigned a constant shade according to that calculation
  - Gouraud shading: an estimate is made of the normal at each vertex of a polygon (for example, by averaging the normals of all surrounding polygons); reflected intensity at each vertex is then calculated; linear interpolation is then used to estimate the intensity at any interior pixel as the shading is carried out
  - Phong shading: as with Gouraud, normal estimates are made at each vertex; but in contrast, we proceed to interpolate the normals at each pixel and then calculate the intensity for that pixel

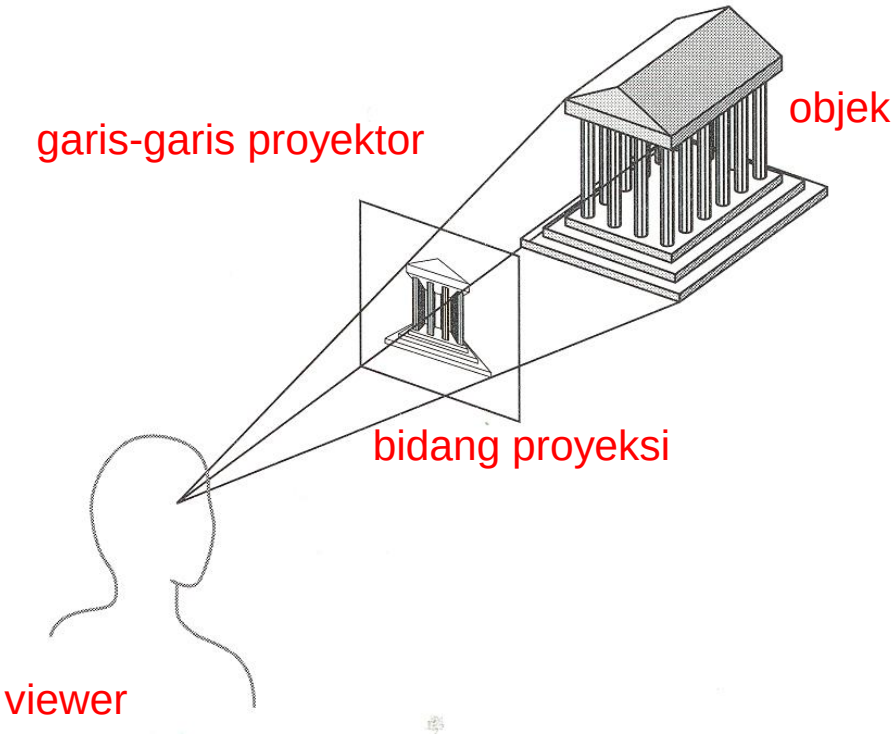
Classical Viewing

Diadaptasi dari Edward Angel's
Interactive Computer Graphics

The University of New Mexico

Classical Viewing

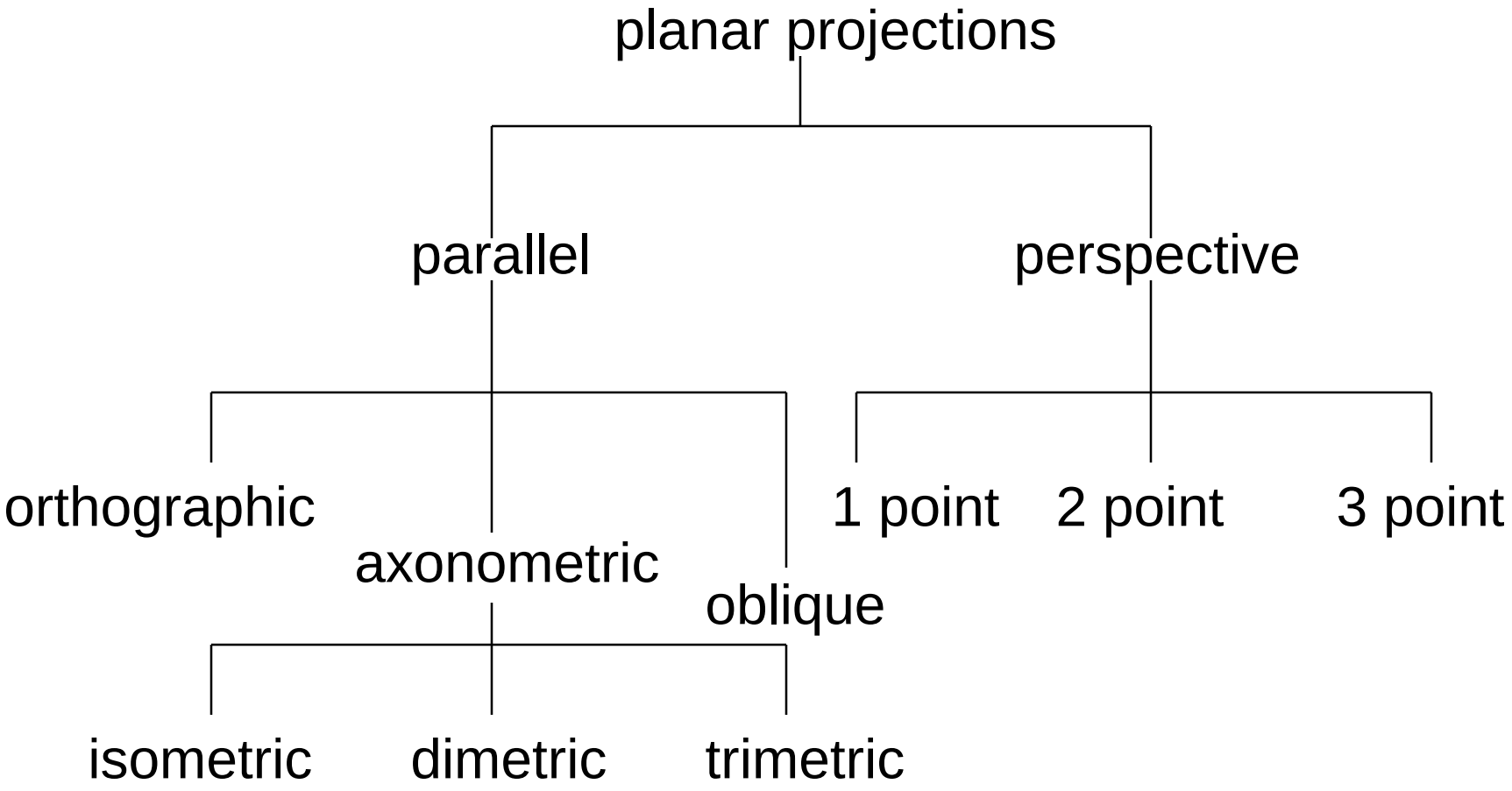
- Viewing terdiri atas:
  - Satu atau lebih objek
  - Viewer dengan suatu bidang proyeksi (projection plane)
  - Proyektor yang memproyeksikan objek ke bidang proyeksi



Proyeksi Planar

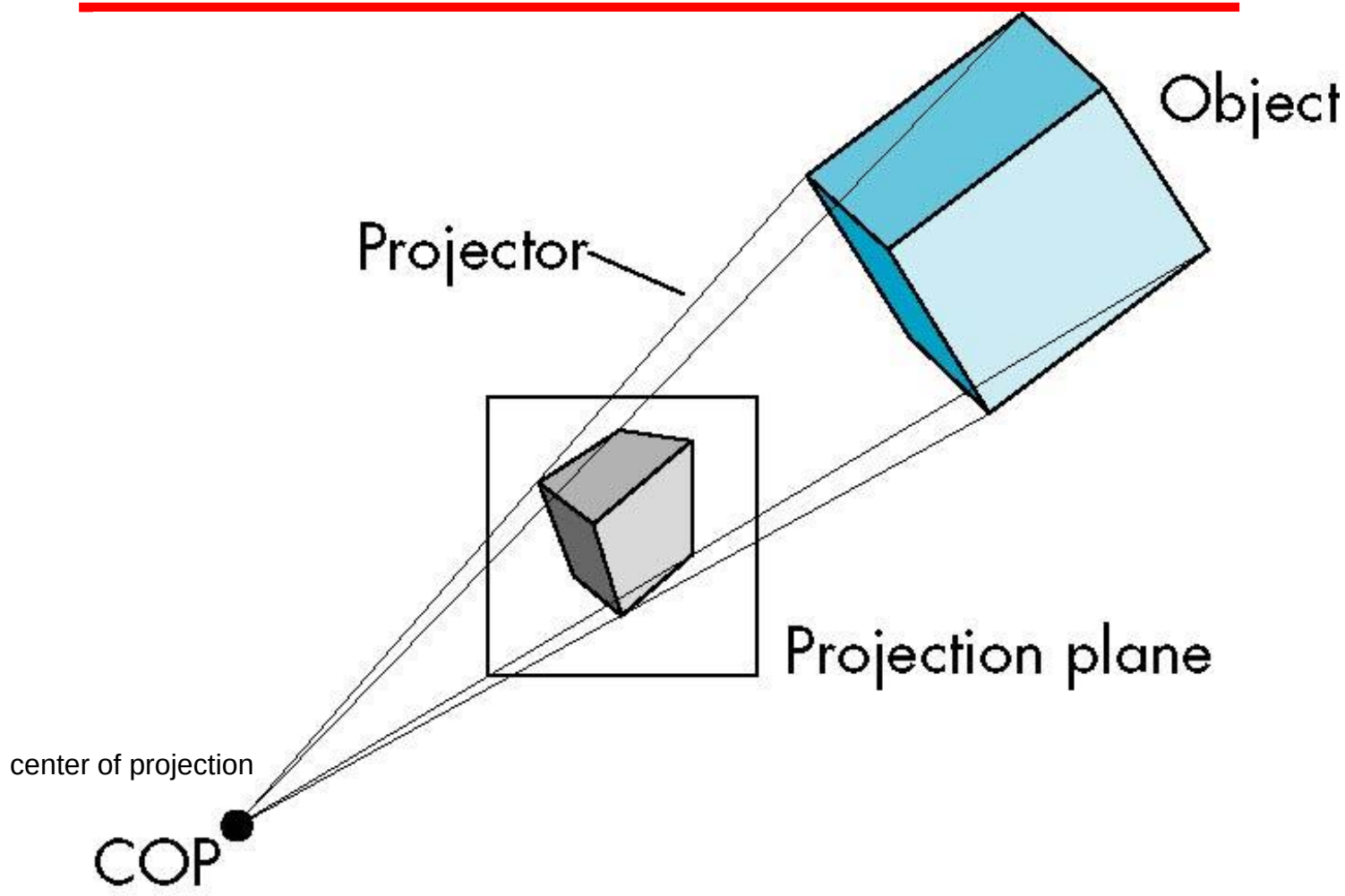
- Proyeksi standar ke suatu bidang datar (planar)
- Proyektor adalah garis yang:
  - bertemu pada satu titik proyeksi, atau
  - sejajar
- Proyeksi menampilkan garis dari objek, tapi tidak selalu menampilkan sudut sesuai aslinya

Kategori Proyeksi Planar

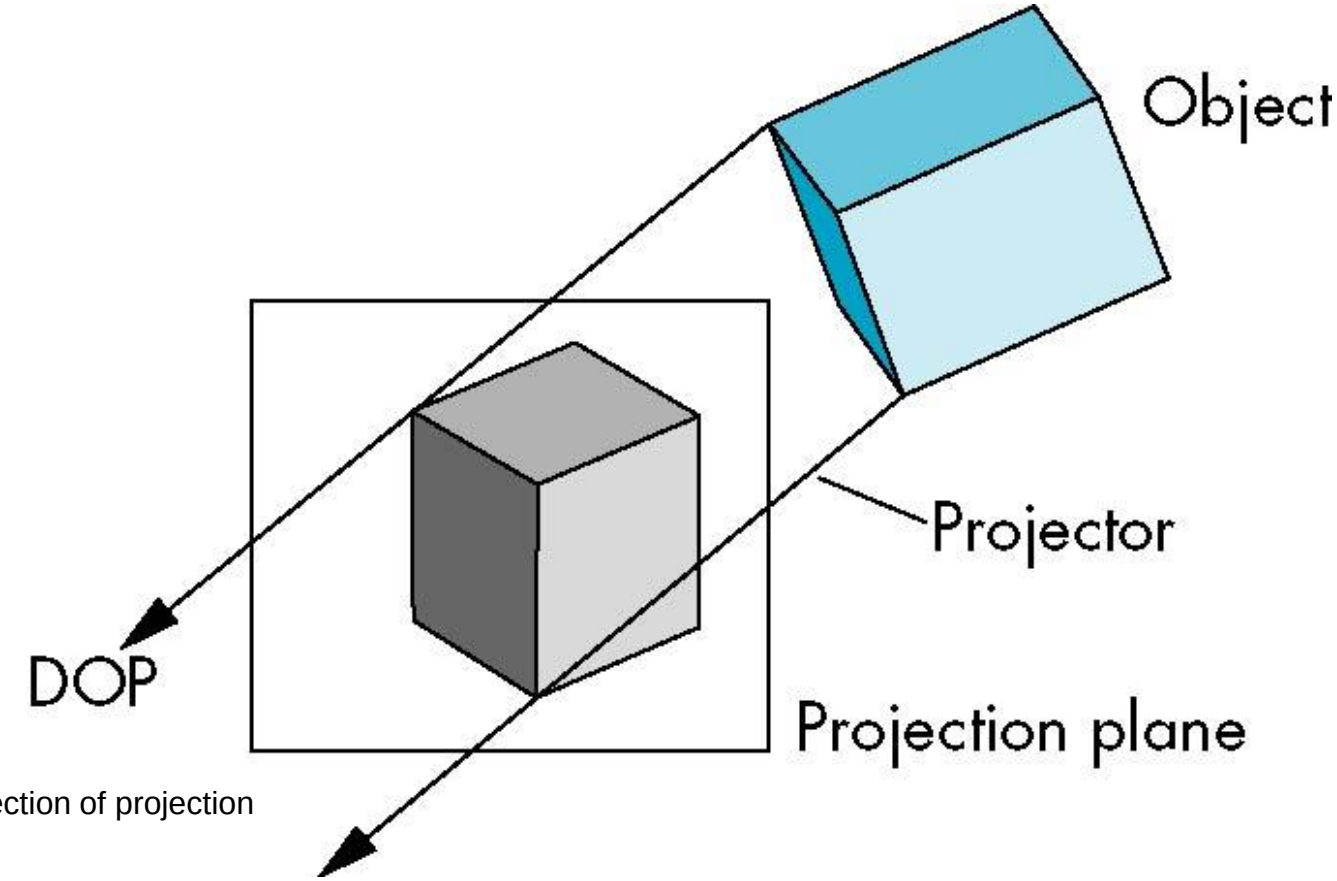


The University of New Mexico

Perspective Projection



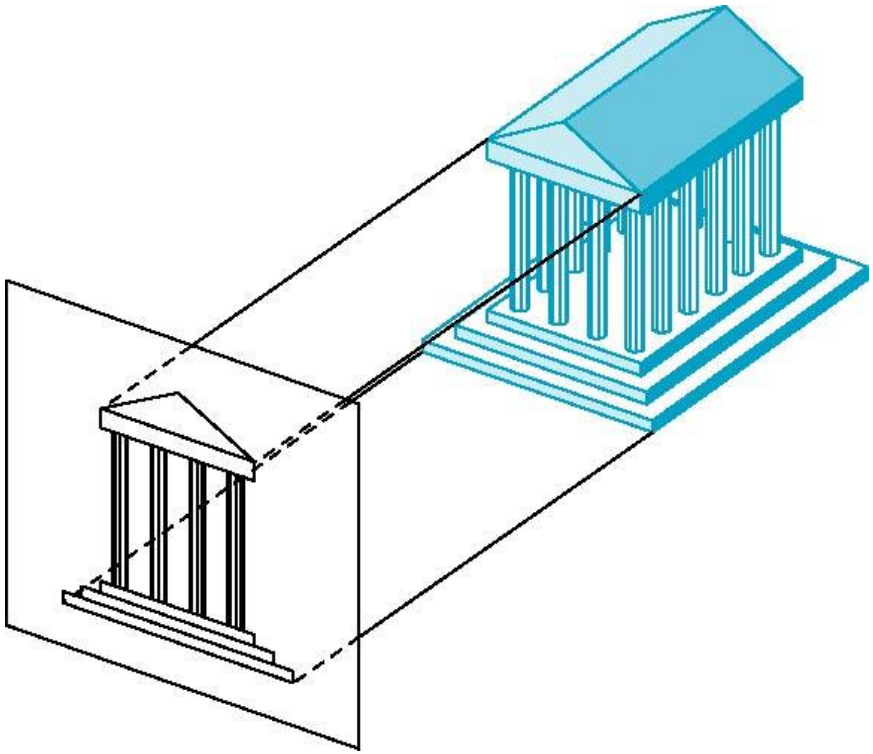
Parallel Projection



The University of New Mexico

Orthographic Projection

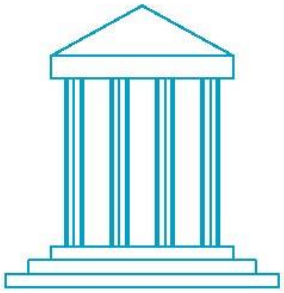
- Garis proyektor tegak lurus (ortogonal) dengan bidang proyeksi



Multiview Orthographic Projection

- Suatu objek dilihat dari beberapa sisi
- Biasanya tampak atas, depan, samping

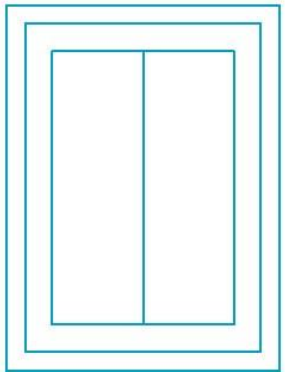
(bukan orthographic)



depan

biasanya digunakan dalam CAD dan arsitektur

atas



samping

Plus dan Minus

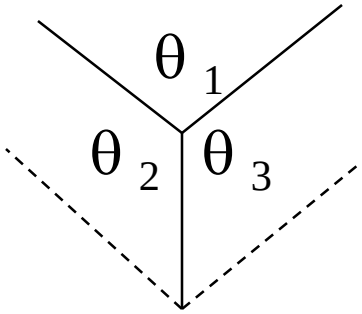
- Mempertahankan jarak dan sudut
  - Bentuk objek dapat dipertahankan
  - Dapat diukur dengan presisi
    - Rancangan bangunan
    - Gambar teknik
- Namun tidak semua sisi dapat ditampilkan
  - Tidak dapat menggambarkan objek secara keseluruhan, apalagi jika objeknya kompleks
  - Dapat disertai dengan proyeksi lainnya

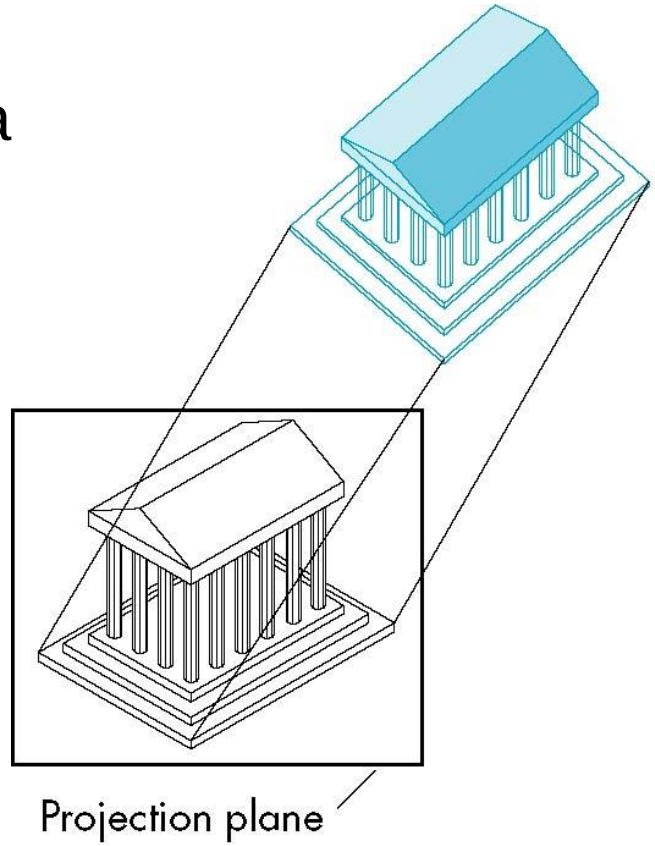
Axonometric Projections

Objek diputar relatif terhadap bidang proyeksi

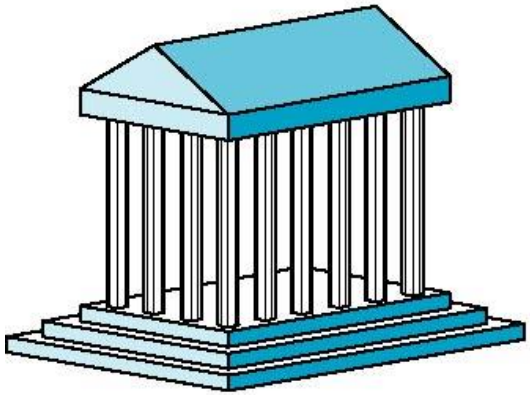
Jenis dibagi berdasarkan berapa banyak sudut yang besarnya sama

nol: trimetric
dua: dimetric
tiga: isometric

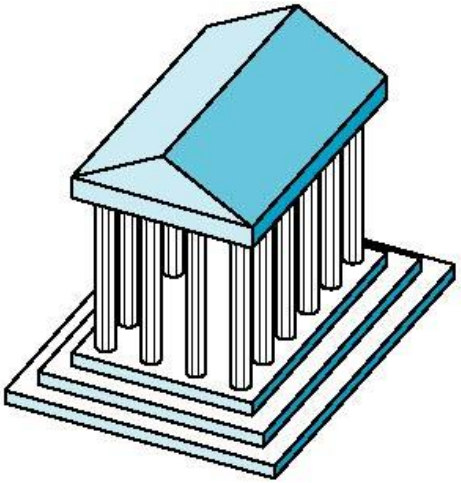




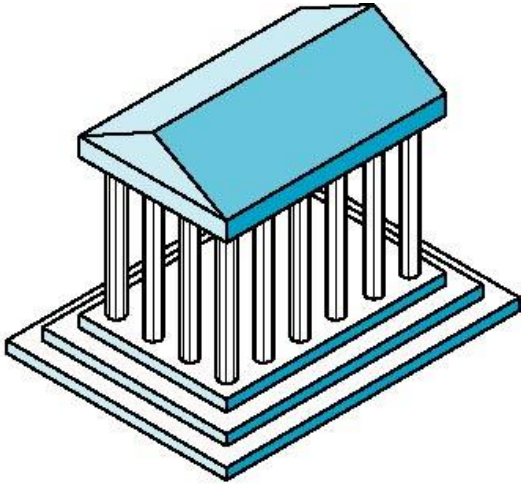
Jenis Axonometric Projections



Dimetric



Trimetric



Isometric

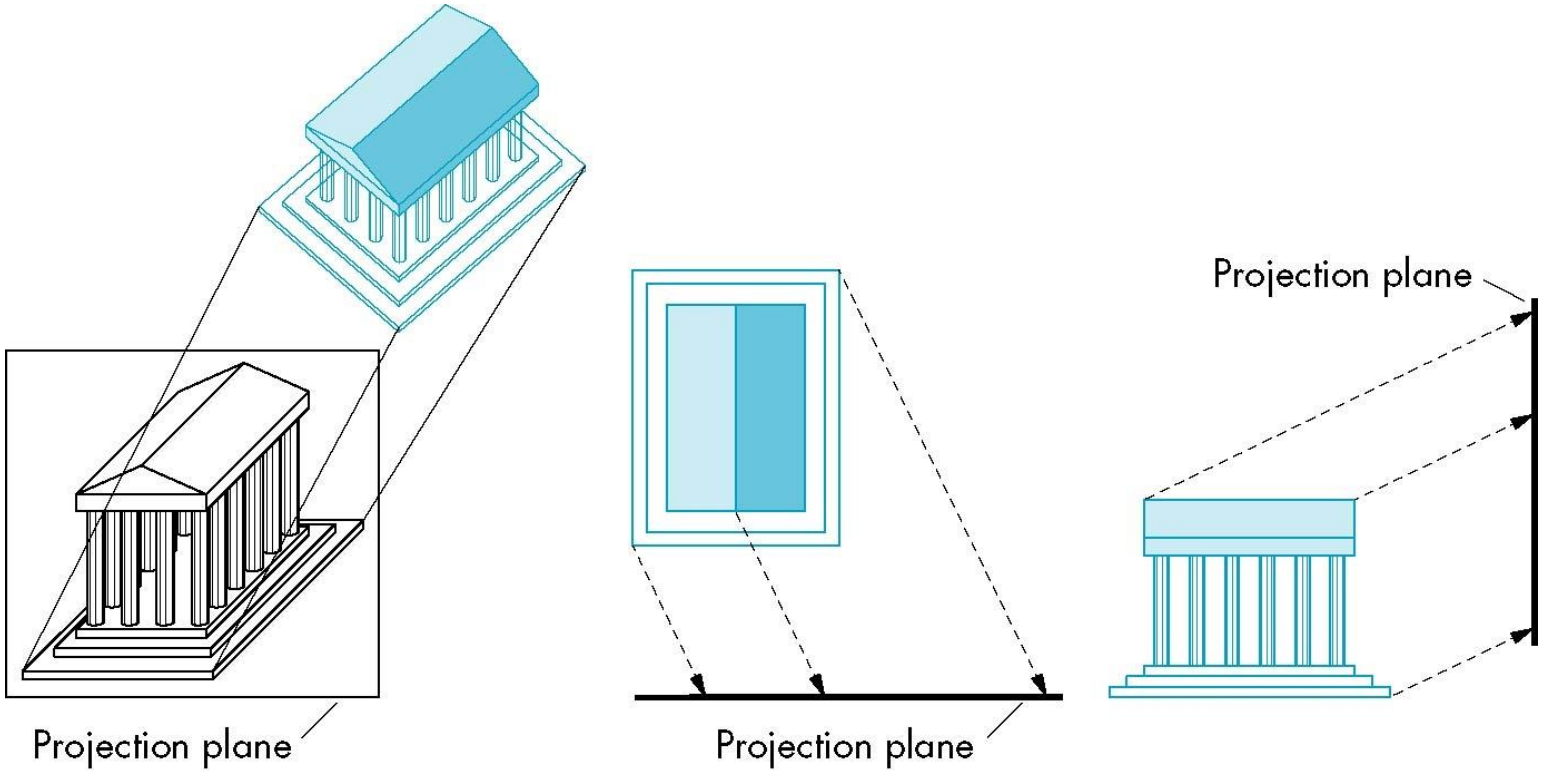
Plus dan Minus

- Garis dari objek dipertahankan, namun sudutnya tidak
- Panjang sisi tidak sama dengan panjang aslinya
- Dapat melihat tiga sisi sekaligus dari suatu objek
- Kurang terlihat realistis karena objek yang jauh ukurannya sama dengan objek yang dekat
- Digunakan juga dalam CAD

The University of New Mexico

Oblique Projection

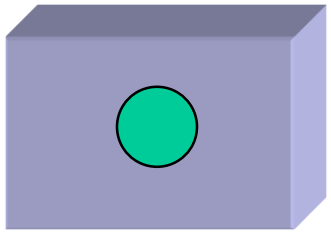
- Proyeksi bebas antara garis proyektor dengan bidang proyeksi



The University of New Mexico

Plus dan Minus

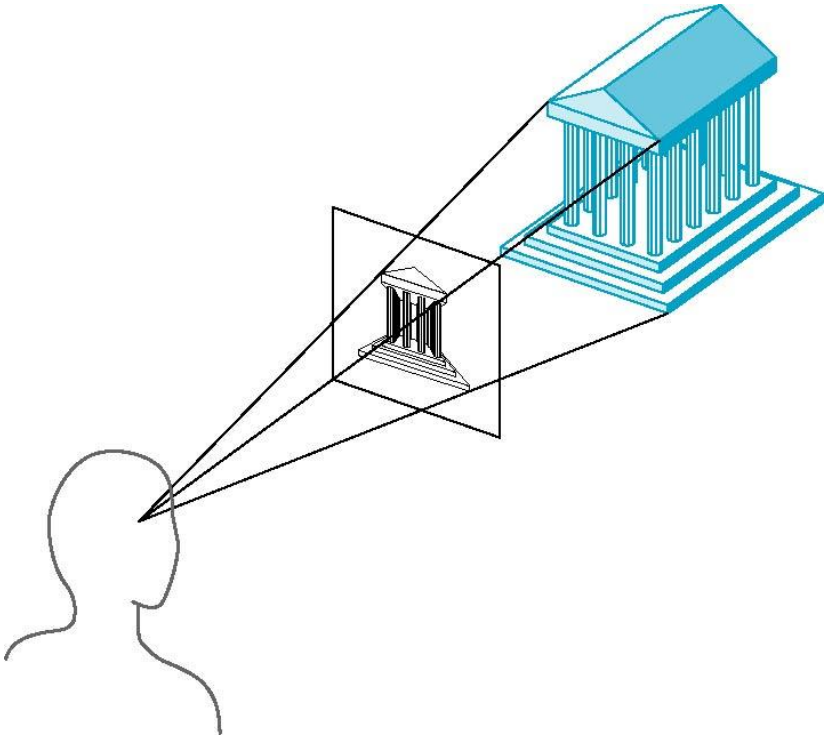
- Dapat digunakan untuk menonjolkan salah satu sisi tertentu
- Seperti orthographic, plus kita juga bisa melihat sisi “samping” dari objek



- Seperti axonometric, kurang realistis

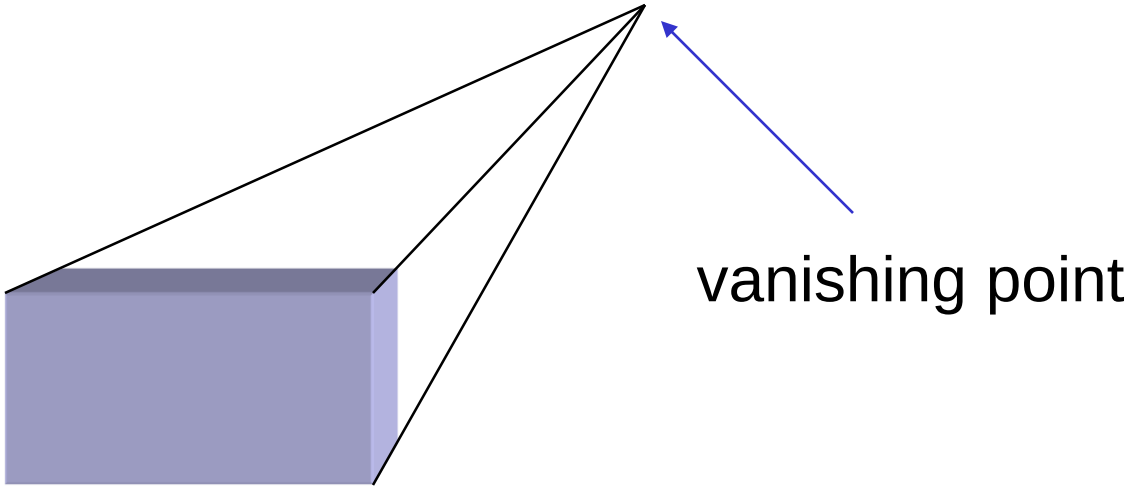
Perspective Projection

Garis proyektor berkumpul pada satu titik



Titik Hilang (Vanishing Point)

- Garis yang sejajar dari objek bertemu pada satu atau lebih vanishing point
- Digunakan juga saat menggambar perspektif dengan tangan



One-Point Perspective

- Satu sisi sejajar bidang proyeksi
- Satu titik hilang untuk kubus

The University of New Mexico

Two-Point Perspective

- Tidak ada sisi yang sejajar bidang proyeksi
- Dua titik hilang untuk kubus

Three-Point Perspective

- Tidak ada sisi yang sejajar bidang proyeksi
- Tiga titik hilang untuk kubus

The University of New Mexico

Plus dan Minus

- Objek yang jauh tampak lebih kecil daripada objek yang dekat
  - Terlihat realistis?
- Ukuran garis juga semakin mengecil jika objek jauh (tidak mempertahankan ukuran)
- Sudut hanya dipertahankan pada sisi yang sejajar bidang proyeksi
- Lebih sulit digambar dengan tangan (tapi tidak bagi komputer)

The University of New Mexico

Contoh dalam Games

Oblique (Earthbound)

The University of New Mexico

Contoh dalam Games

One-Point Perspective (Grand Theft Auto 1)

The University of New Mexico

Contoh dalam Games

Isometric (Age of Empires II)

The University of New Mexico

Contoh dalam Games

Trimetric (SimCity 4)

The University of New Mexico

Contoh dalam Games

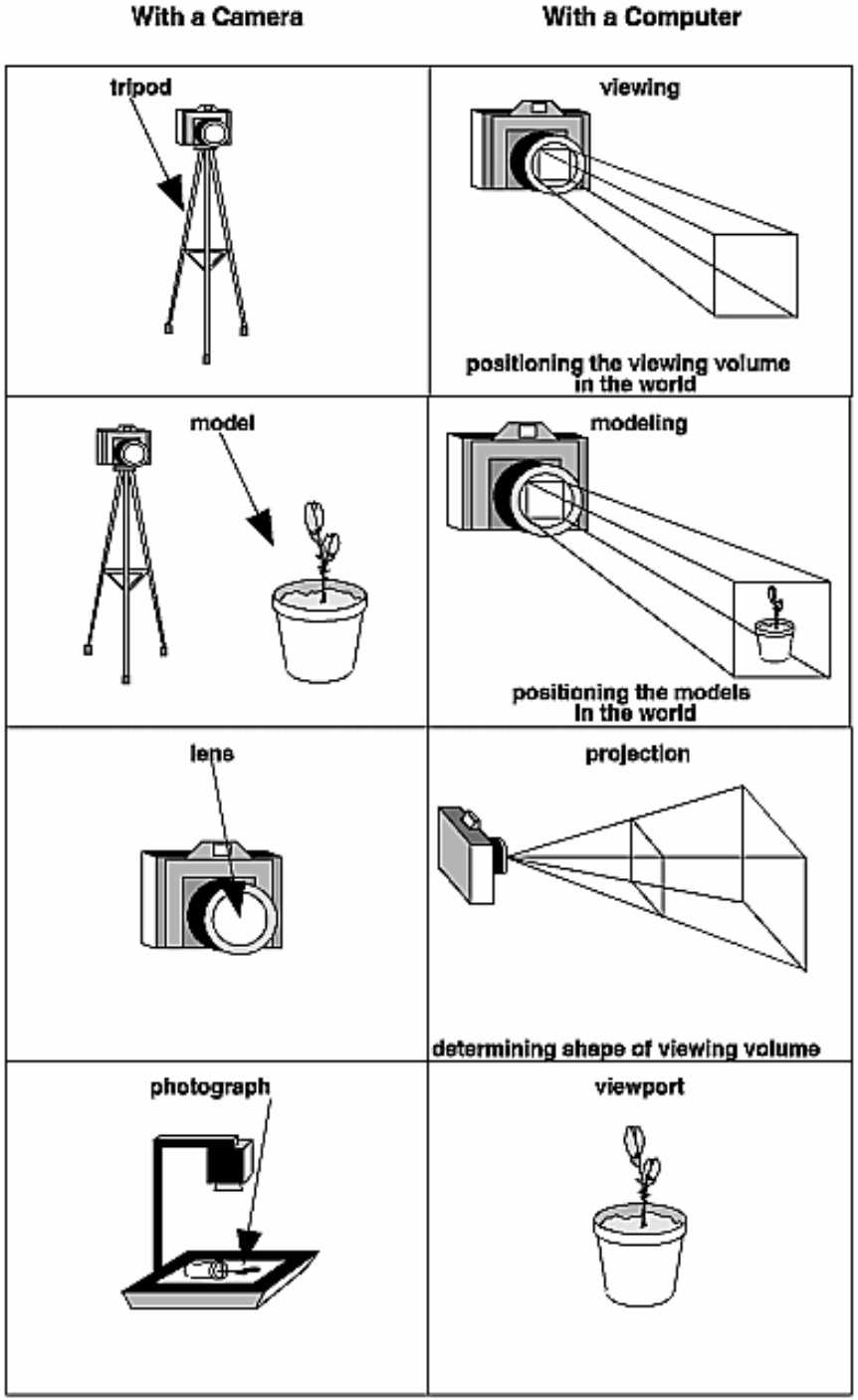
Three-Point Perspective (Age of Empires III)

Computer Viewing

Computer Viewing

- Analogi seperti memotret dengan kamera
- Terdapat tiga proses pada pipeline:
  - Memposisikan “kamera”
    - Setting matriks model-view
  - Memilih “lensa”
    - Setting matriks projection
  - Clipping
    - Setting view volume

Analogi dengan Kamera



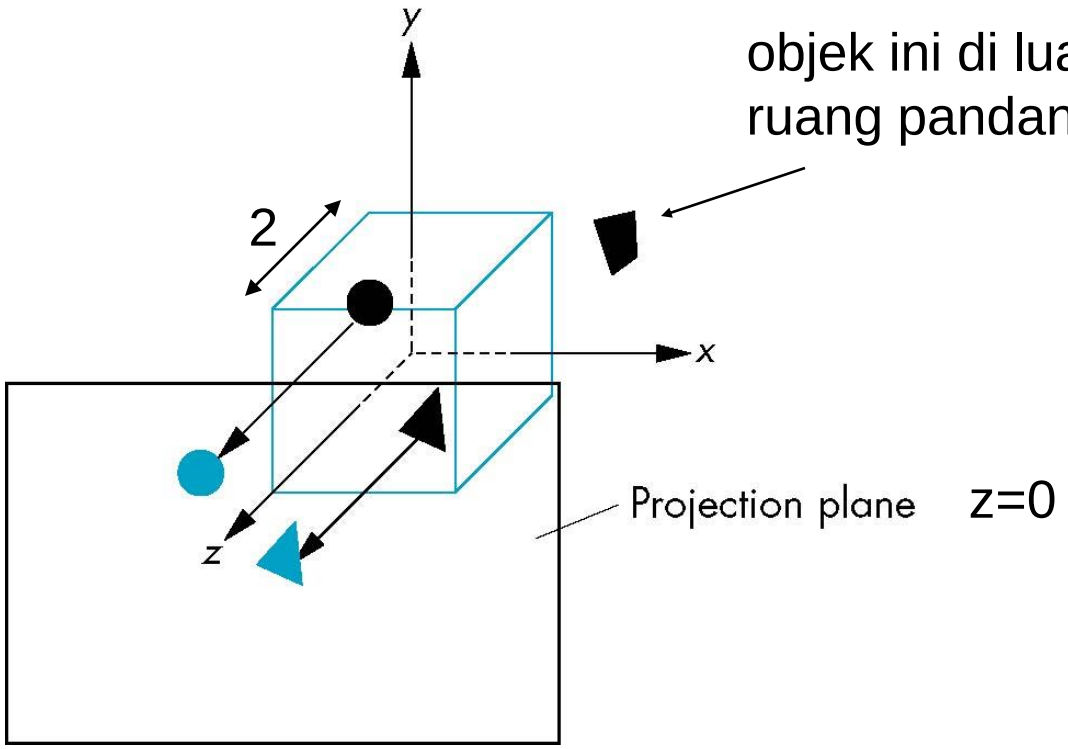
OpenGL Camera

- Matriks model-view dan projection default adalah matriks identitas (`glLoadIdentity`)
- Secara default, kamera terletak di pusat koordinat (0, 0, 0) dan mengarah ke sumbu Z negatif
- Ruang pandang default adalah suatu kubus di pusat koordinat dengan panjang sisi 2 satuan

The University of New Mexico

Default Projection

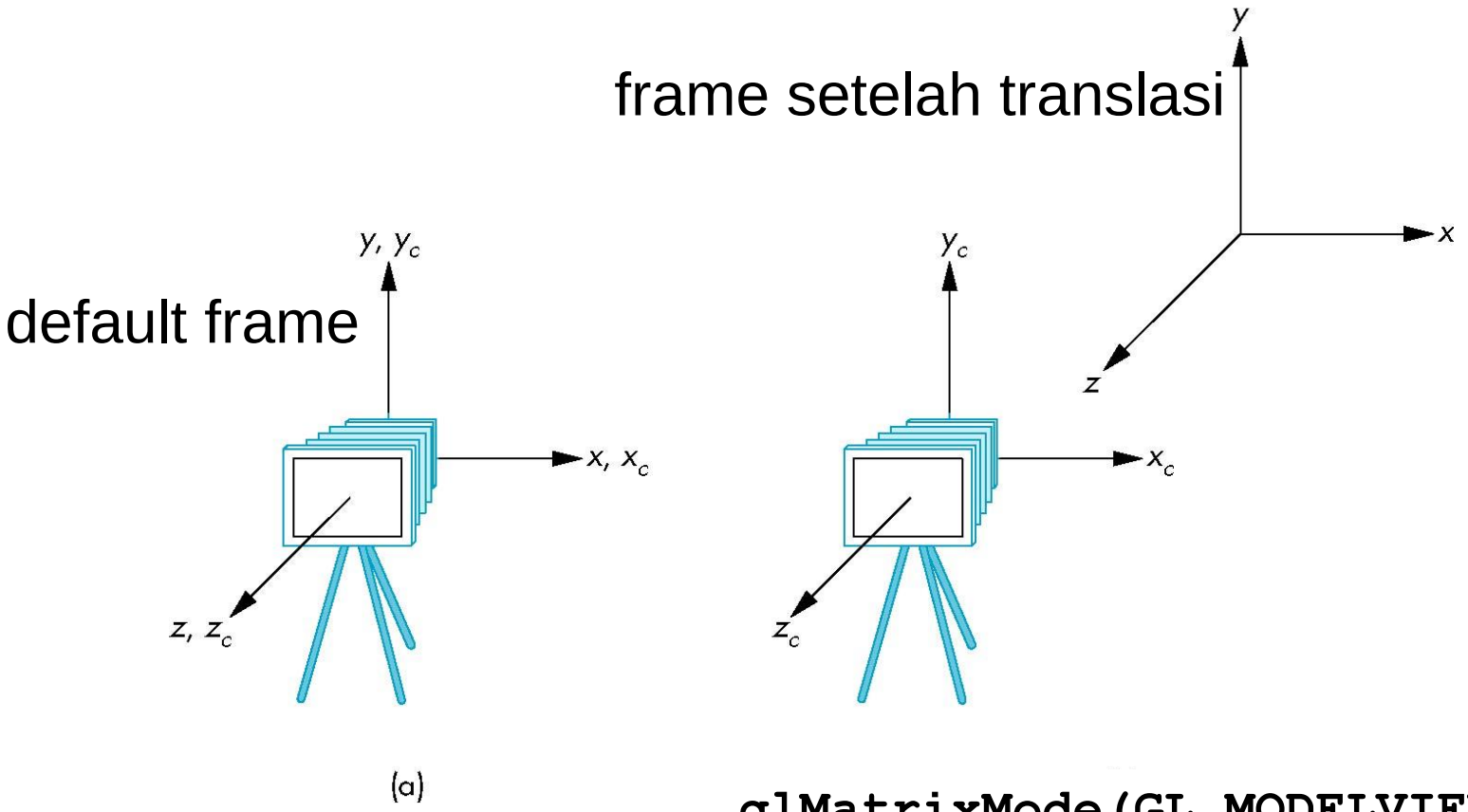
Default projection adalah orthographic



Memindahkan Camera Frame

- Jika kita ingin melihat objek dengan nilai Z positif:
  - Pindahkan kamera ke arah Z positif (translasikan camera frame), atau
  - Pindahkan objek ke arah Z negatif (translasikan world frame)
- Kedua alternatif tersebut sama saja dalam hal manipulasi matriks model-view

Menggeser Camera dari Pusat




```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity();
glTranslatef(0.0, 0.0, -d);
```

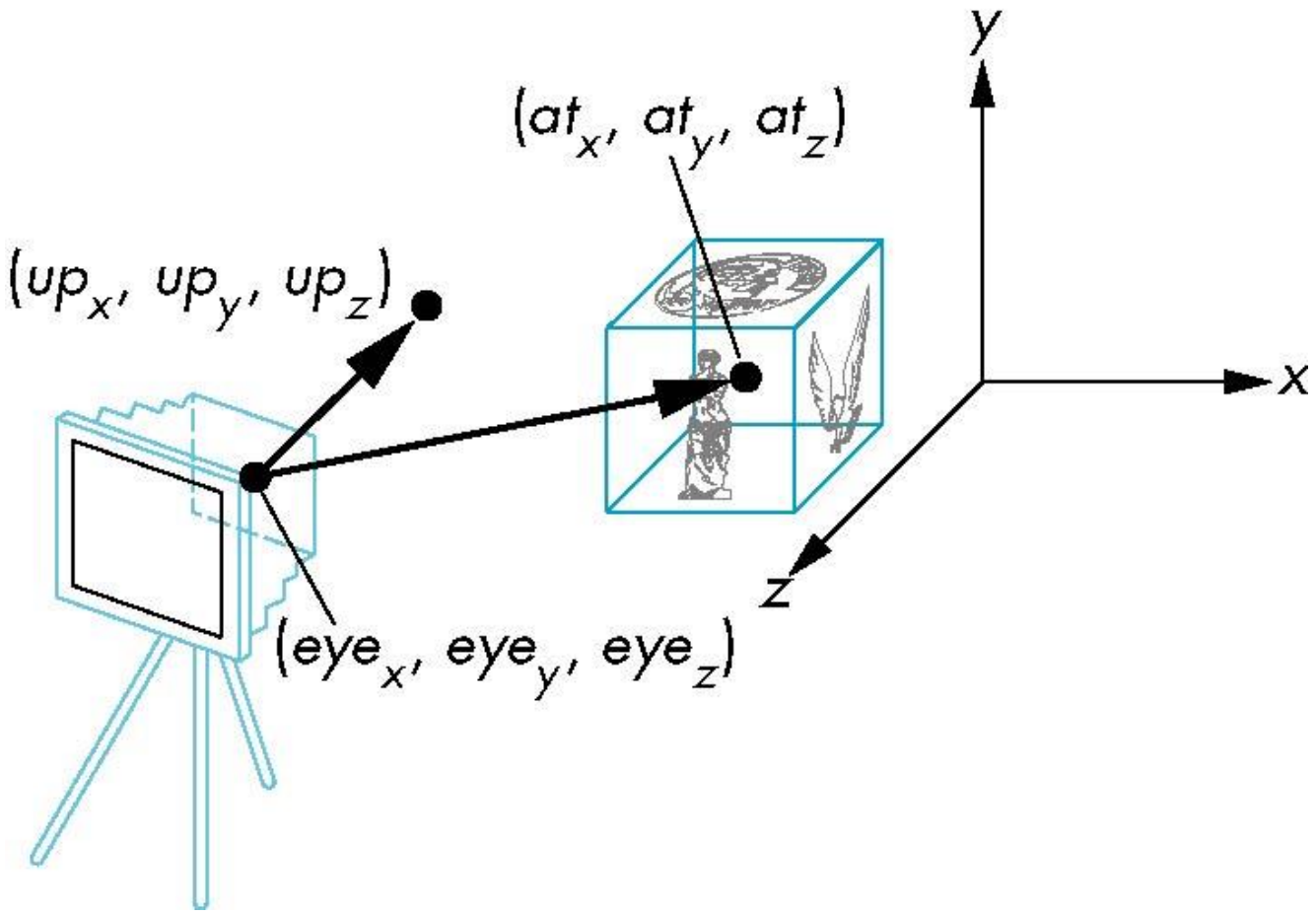
Fungsi LookAt

- Pustaka GLU menyediakan fungsi `gluLookAt` untuk manipulasi matriks model-view
- Perlu mengeset “up vector”
- Contoh: tampilan isometric:

```
glMatrixMode(GL_MODELVIEW):
glLoadIdentity();
gluLookAt(1.0, 1.0, 1.0,
          0.0, 0.0, 0.0,
          0.0, 1.0. 0.0);
```

gluLookAt

```
glLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
```



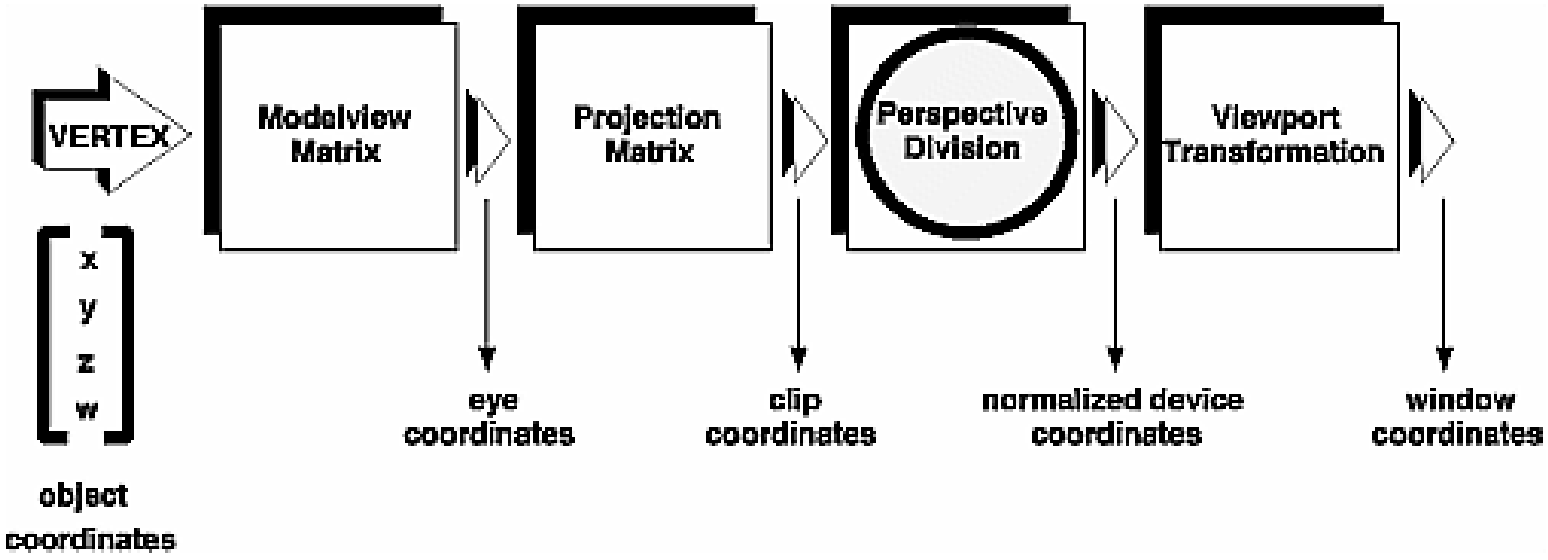
DONE

Computer Viewing

Projection and Viewport Transformation

The University of New Mexico

Pipeline



Proyeksi dan Normalisasi

- Proyeksi default adalah ortogonal
- Selain proyeksi, dilakukan juga normalisasi
  - Semua jenis viewing dikonversi menjadi *view volume* default berdasarkan matriks proyeksinya

Koordinat Homogen

proyeksi ortogonal default:

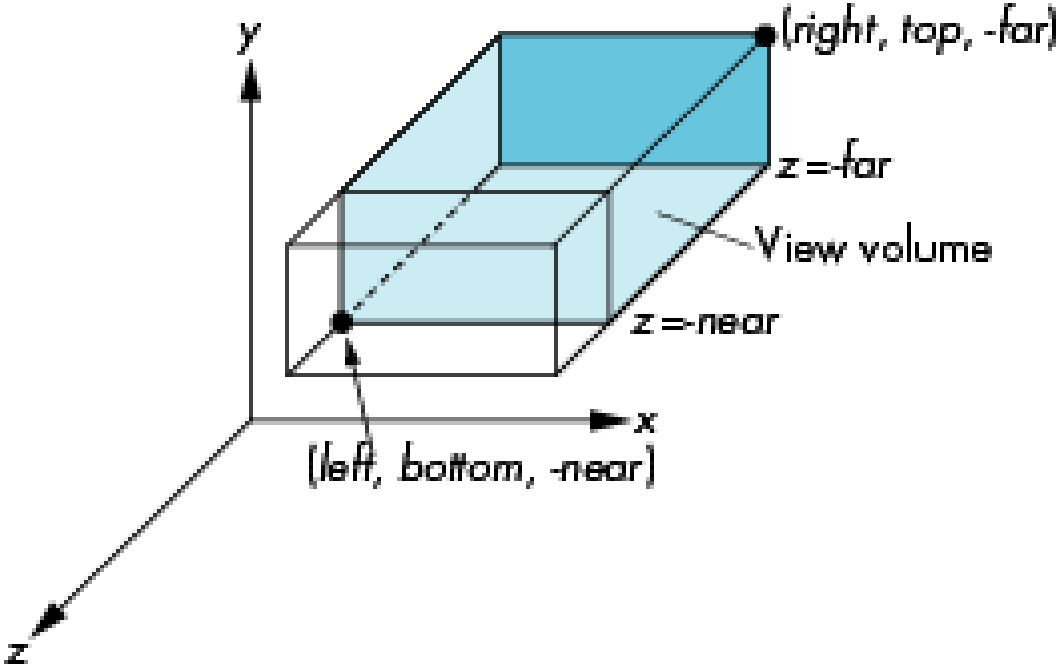
$$p_p = M \times p$$

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
glLoadIdentity()
```

OpenGL Orthogonal Viewing

```
glOrtho(left,right,bottom,top,near,far)
```



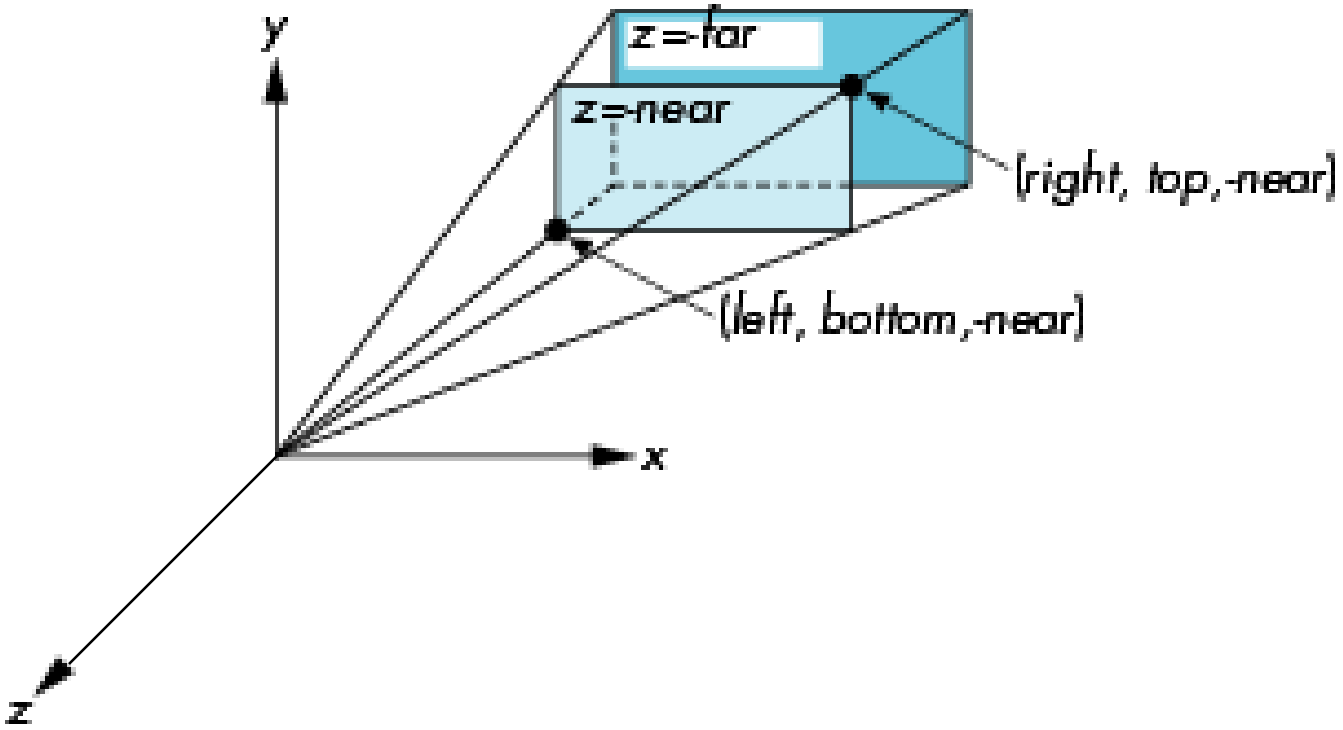
Matriks Proyeksi Ortogonal

matriks proyeksi ortogonal dengan parameter left, right, bottom, top, near, far:

$$M = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

OpenGL Perspective

```
glFrustum(left,right,bottom,top,near,far)
```



Matriks Proyeksi Perspektif

frustum proyeksi dengan parameter left, right, bottom, top, near, far:

$$M = \begin{bmatrix} \frac{2 \cdot near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2 \cdot near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & -\frac{2 \cdot far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Contoh

```
glFrustum(-1, 1, -1, 1, 2, 10)
```

$$M = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & -1.5 & -5 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Misalnya $p = \begin{bmatrix} 1 \\ 1 \\ -2 \\ 1 \end{bmatrix}$, berapa nilai setelah proyeksi (p_p)?

Perspective Division

- Perkalian dengan matriks perspektif menghasilkan nilai $w \neq 1$
- OpenGL membagi nilai (x, y, z) dengan w agar kembali ke koordinat homogen

$$p_p = \begin{bmatrix} 2 \\ 2 \\ -2 \\ 2 \end{bmatrix} \xrightarrow{M} p_p = \begin{bmatrix} 2 \\ 2 \\ -2 \\ 2 \end{bmatrix} / 2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

Perspective Division

- Pembagian dengan w membuat titik dengan nilai -z lebih besar menjadi lebih ke “tengah”

$$p = \begin{bmatrix} 1 \\ 1 \\ -2 \\ 1 \end{bmatrix} \xrightarrow{M} p_p = \begin{bmatrix} 2 \\ 2 \\ -2 \\ 2 \end{bmatrix} /2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

$$p = \begin{bmatrix} 1 \\ 1 \\ -3 \\ 1 \end{bmatrix} \xrightarrow{M} p_p = \begin{bmatrix} 2 \\ 2 \\ -0.5 \\ 3 \end{bmatrix} /3 = \begin{bmatrix} 0.66 \\ 0.66 \\ -0.16 \\ 1 \end{bmatrix}$$

Field of View

- **`glFrustum`** terkadang sulit menentukan viewing yang “pas”
- **`gluPerspective(fov, aspect, near, far)`** menyediakan alternatif yang lebih mudah
- Konversi parameter ke **`glFrustum`**:

$$top = near \cdot \tan\left(\frac{\pi}{180} \cdot FOV/2\right)$$

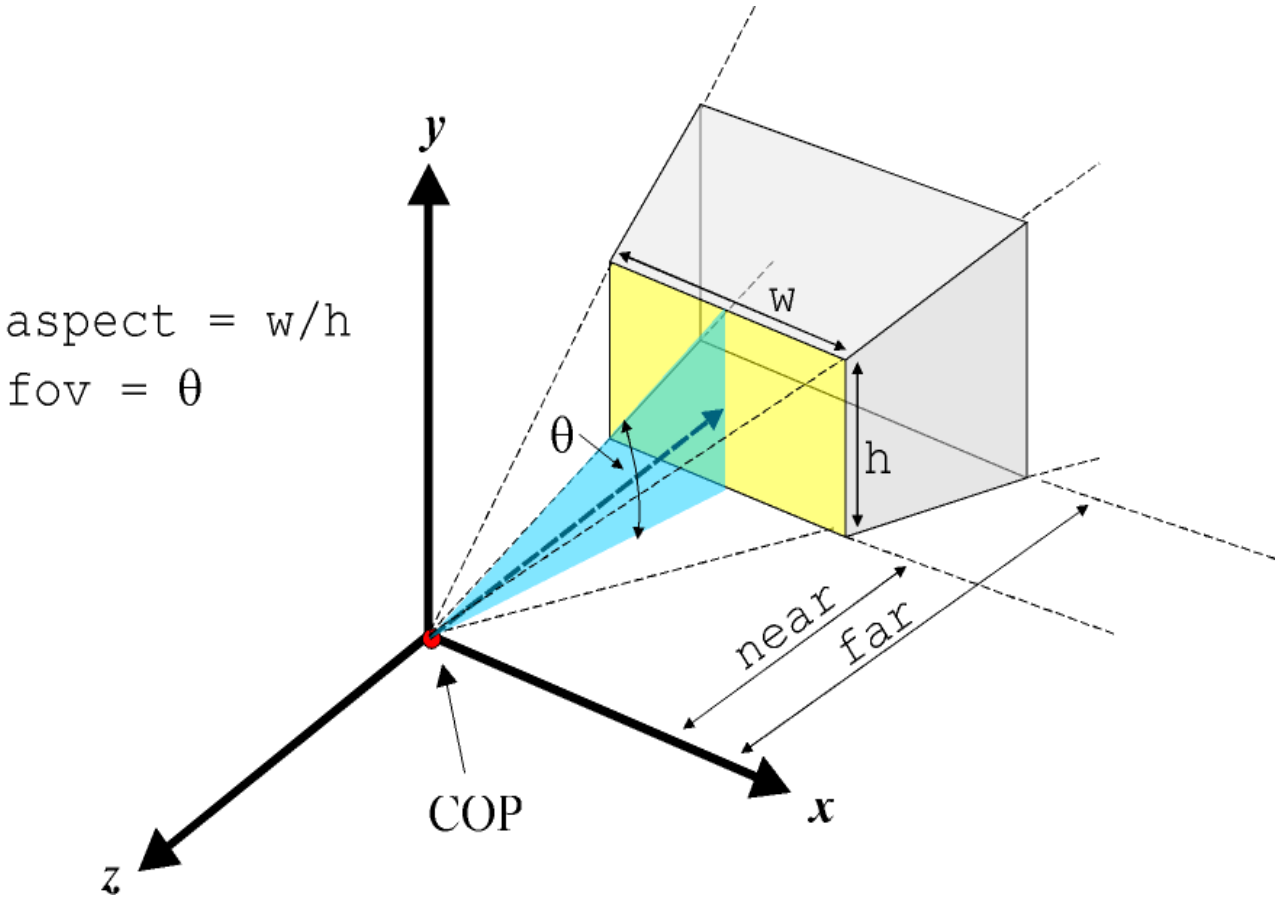
$$bottom = -top$$

$$right = top \cdot aspect$$

$$left = -right$$

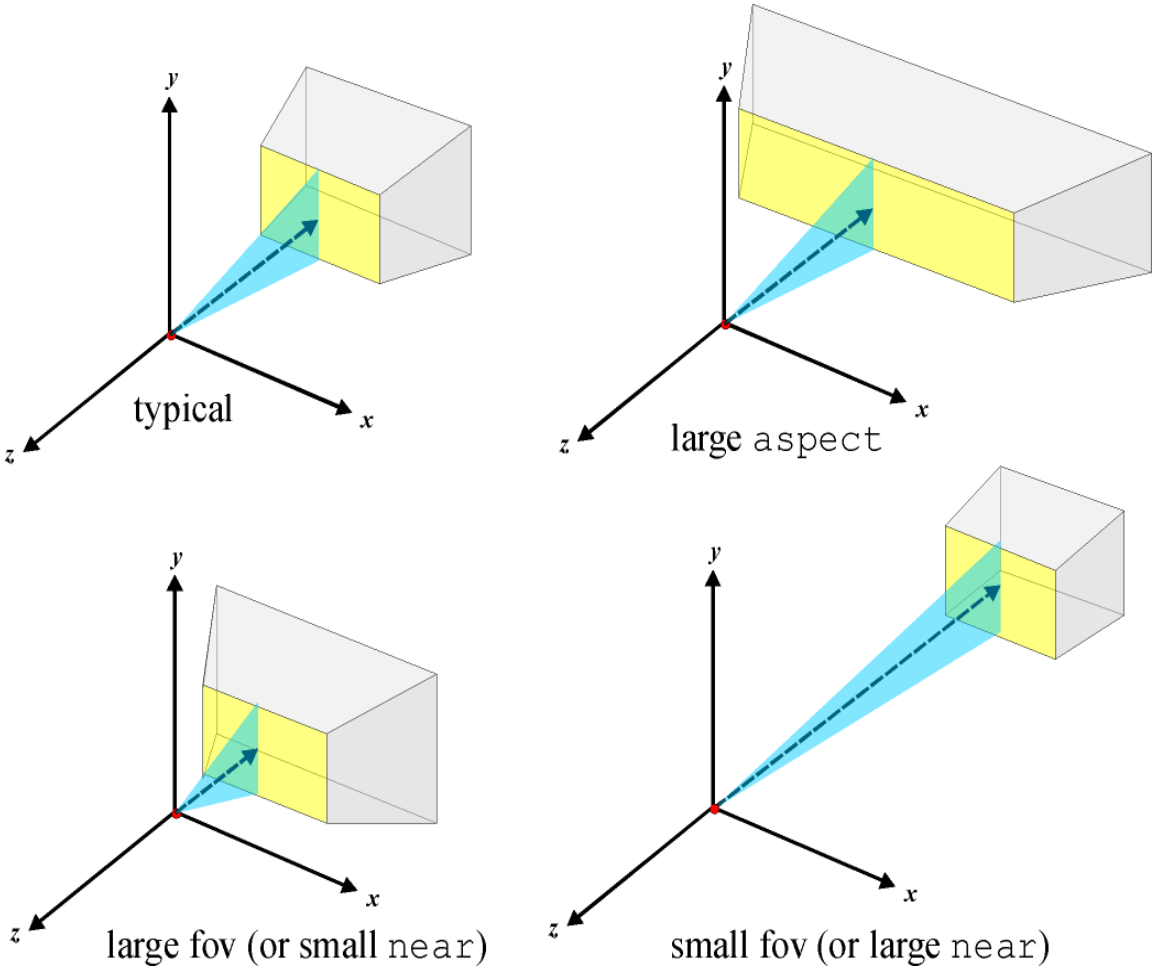
The University of New Mexico

gluPerspective(fov, aspect, near, far)



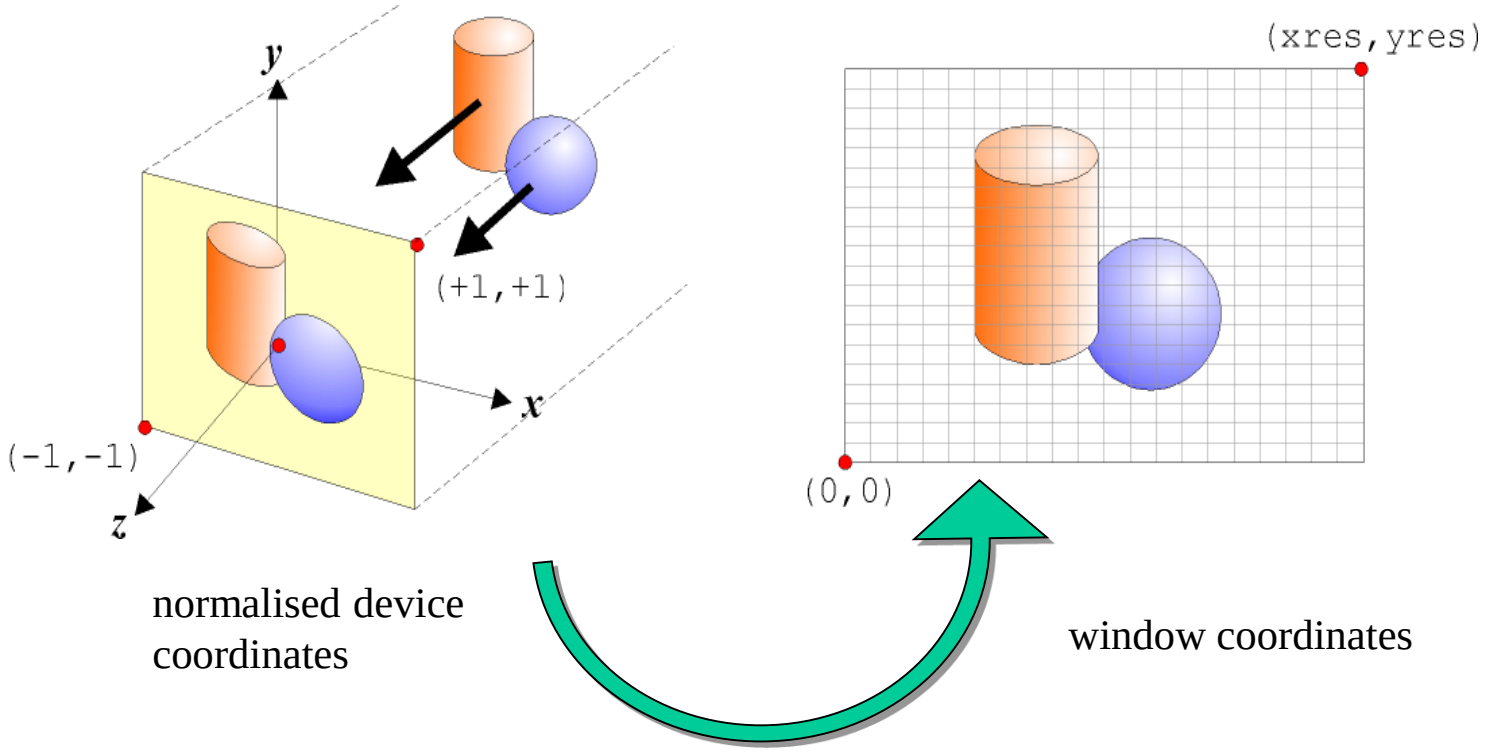
The University of New Mexico

Perspectives



Viewport Transformation

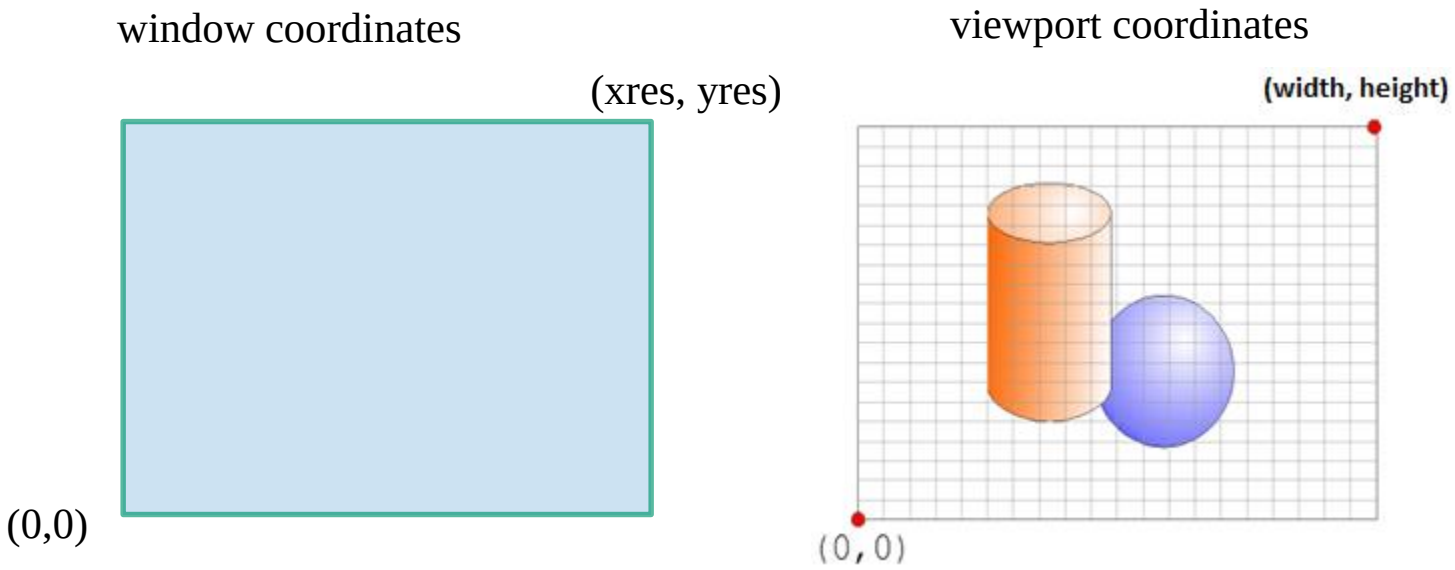
- Pemetaan koordinat ke window / monitor



glViewport

```
glViewport(x, y, width, height)
```

- koordinat (x,y) window = lokasi kiri bawah viewport
- `width, height` = ukuran viewport (dalam pixel)

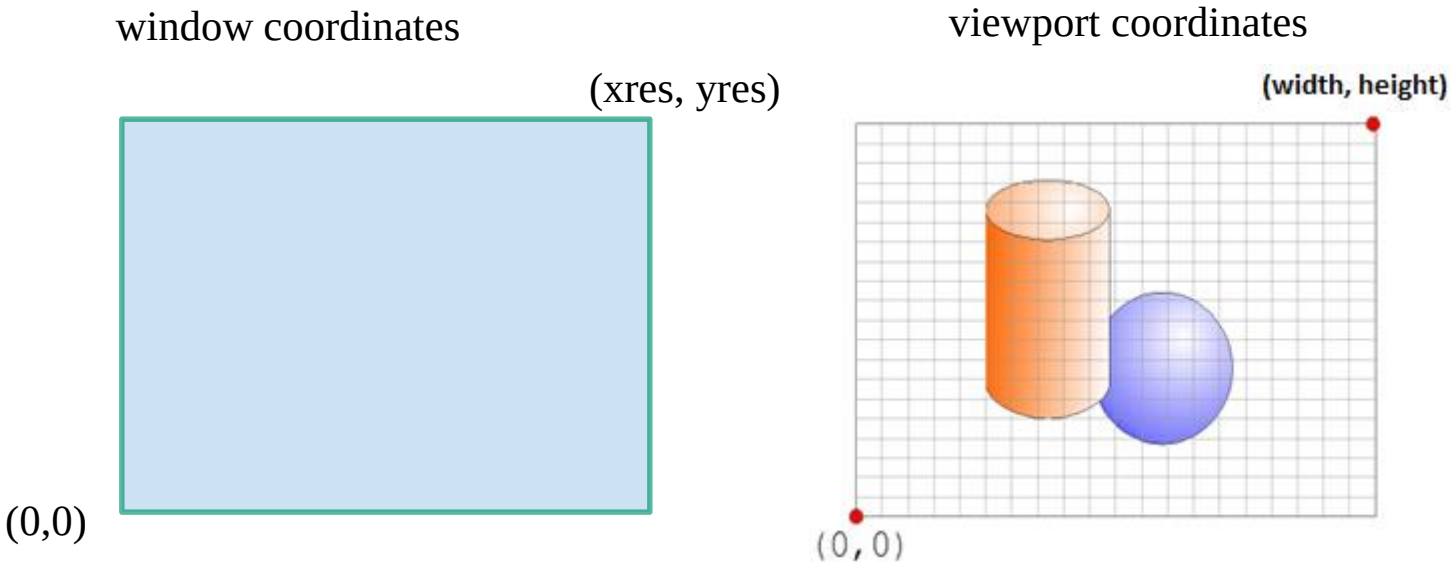


The University of New Mexico

glViewport

```
width = xres; height = yres;
glViewport(0, 0, width, height);
```

- Titik kiri bawah bidang viewport berada pada koordinat (0,0) window, ukuran viewport sama dengan ukuran window.

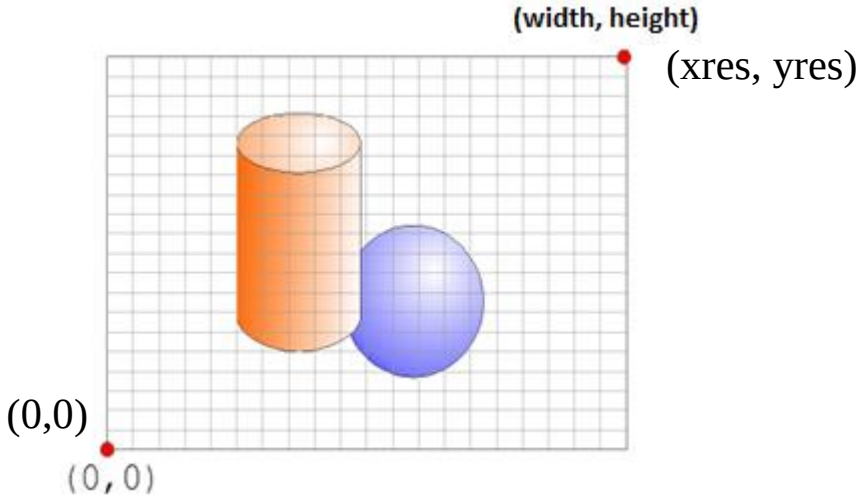


glViewport

```
width = xres; height = yres;
glViewport(0, 0, width, height);
```

- Titik kiri bawah bidang viewport berada pada koordinat (0,0) window, ukuran viewport sama dengan ukuran window.

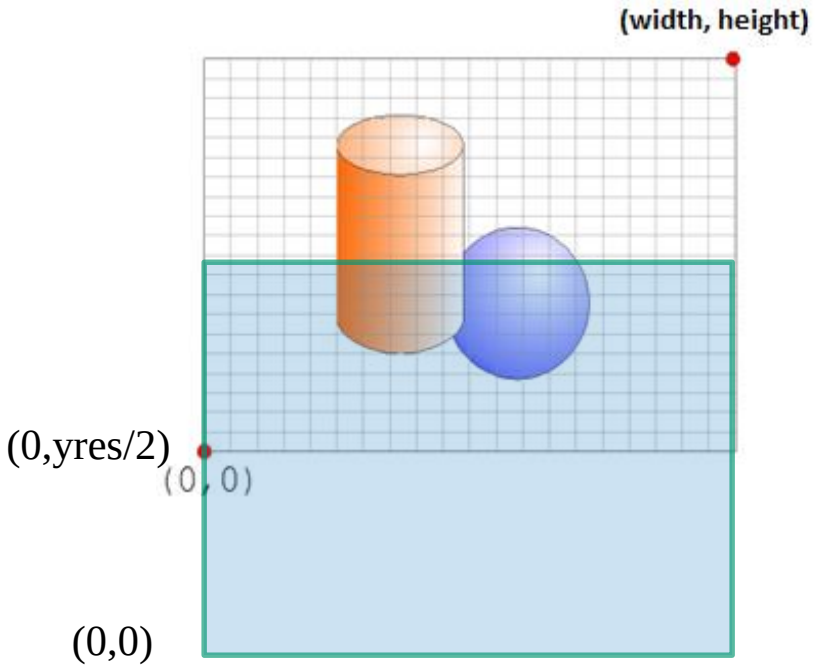
window & viewport coordinates



glViewport

```
width = xres; height = yres;
glViewport(0, yres/2, width, height);
```

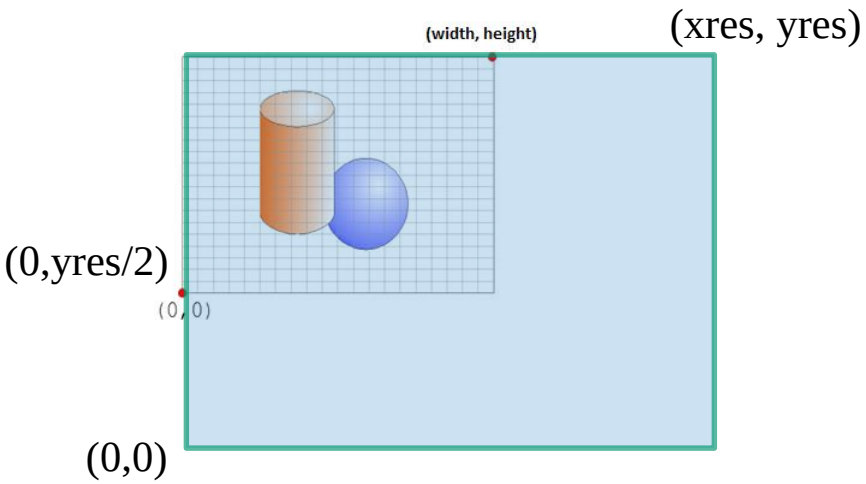
- Hanya setengah bagian dari viewport yang terlihat di layar



glViewport

```
width = xres/2; height = yres/2;
glViewport(0, yres/2, width, height);
```

- Titik kiri bawah bidang viewport berada pada koordinat (0, yres/2) window, dengan bidang viewport separuh lebar dan tinggi window



DONE