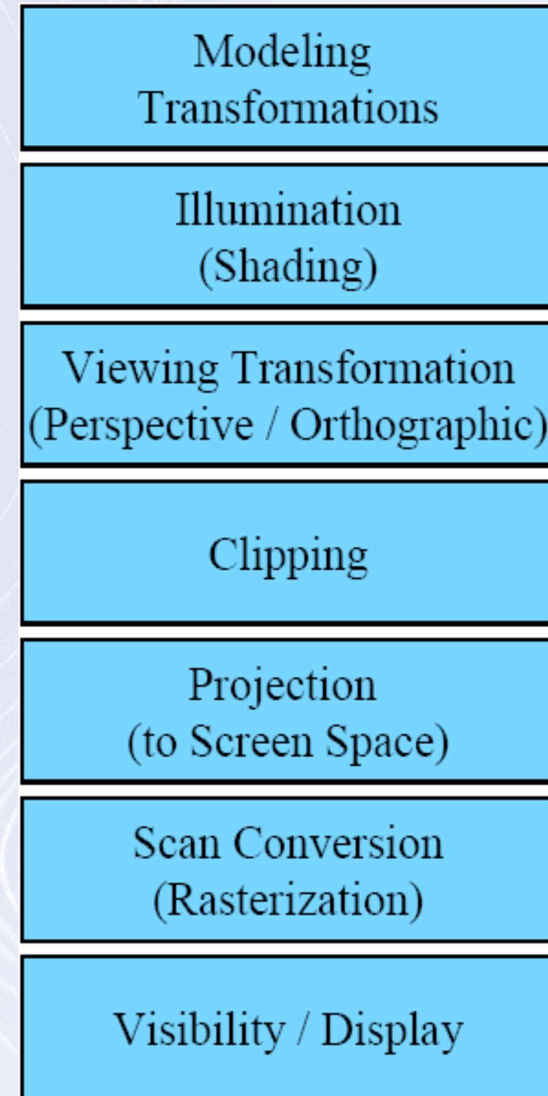
3D Computer Graphics with OpenGL and JOGL

An introduction

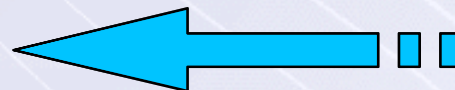
Véronique GAILDRAT
Patrice TORGUET
VORTEX/IRIT/UPS

Graphic Pipeline

- Input:
  - Geometric Model
  - Lighting and Shading Models
  - Camera Models
  - Viewport

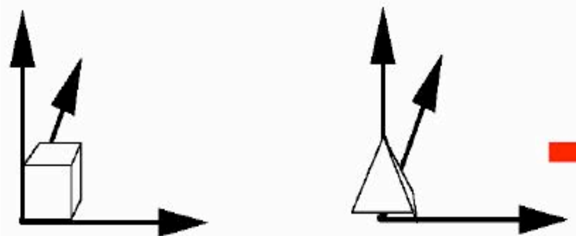


- Output:
  - A color value for each pixel in the image memory

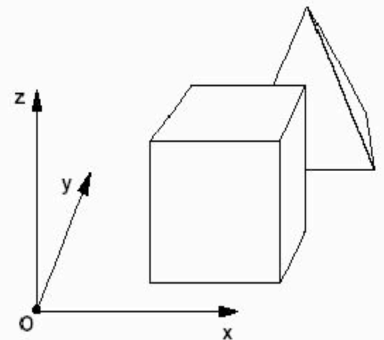


Graphic Pipeline

- 3D Objects:
  - Are defined in their own coordinate system (Local CS)
  - Giving a position to the 3D Object = computing a coordinate system transformation (Transform Local CS into the World CS)



Local CS



World CS

Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

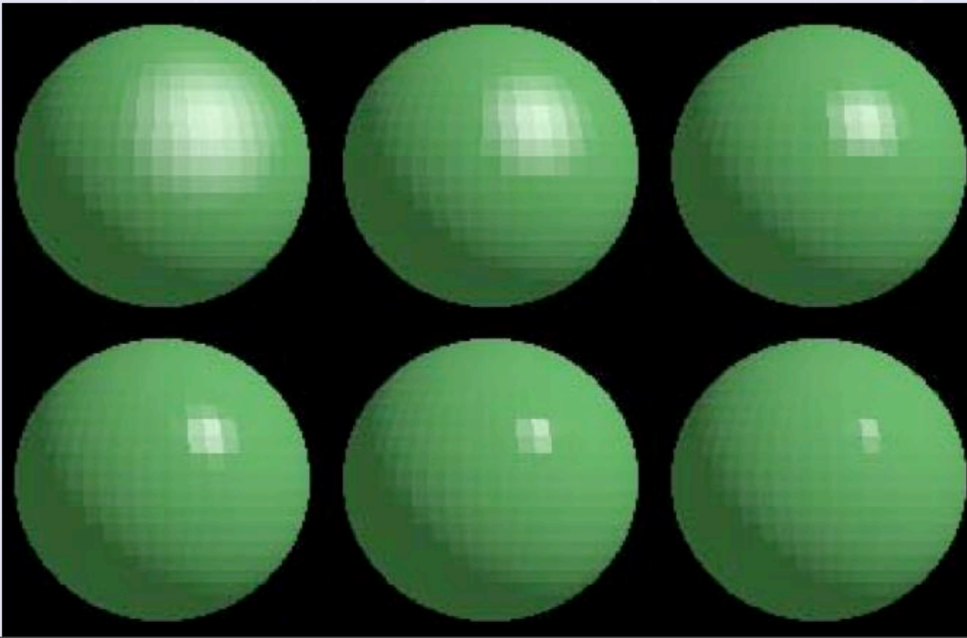
Projection (to Screen Space)

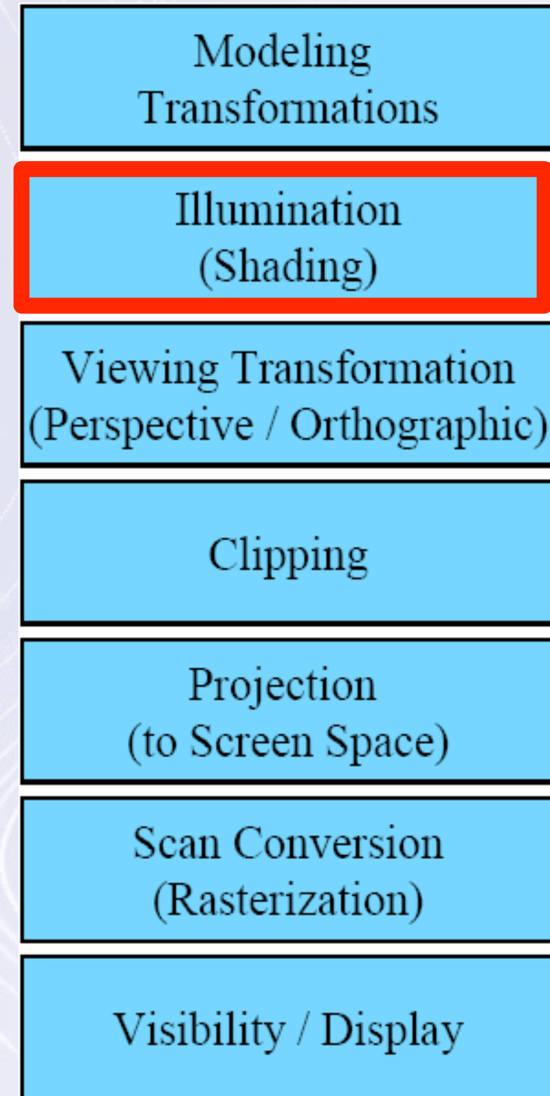
Scan Conversion (Rasterization)

Visibility / Display

Graphic Pipeline

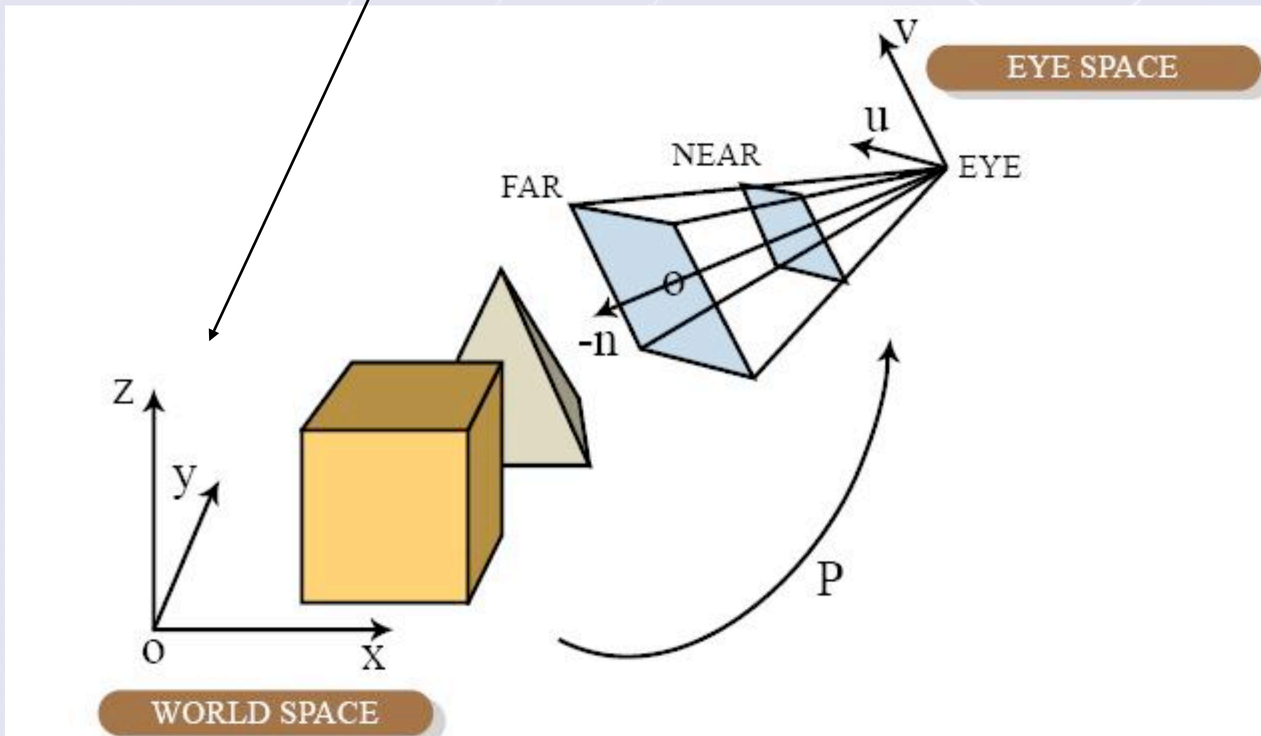
- Lighting based on:
  - Material properties
  - Surface properties (normal vectors)
  - Light sources
- Local Shading Models:
  - (diffuse, ambient, Gouraud, Phong, etc.)





Graphic Pipeline

- Transform World CS into the camera local CS



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

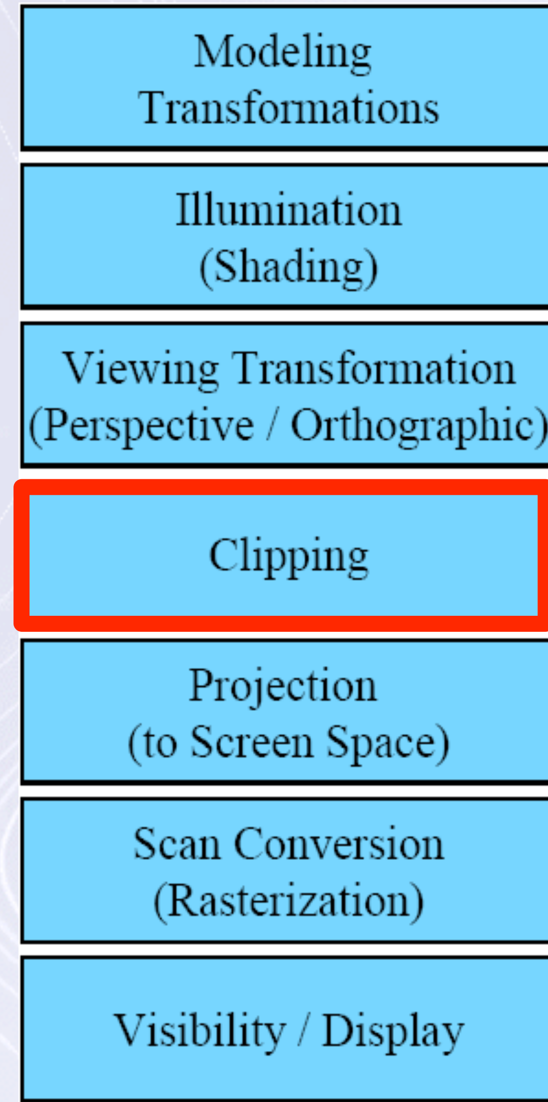
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

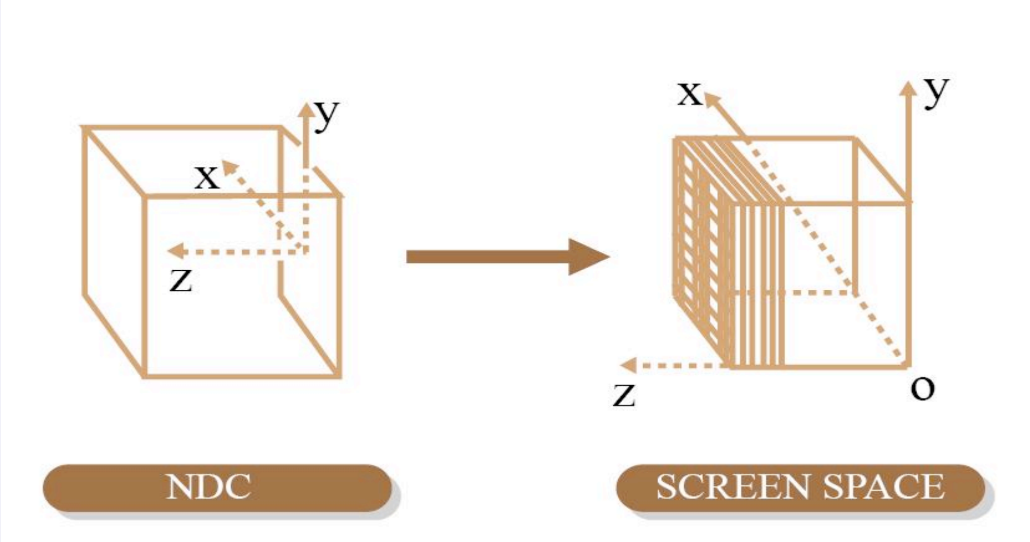
Graphic Pipeline

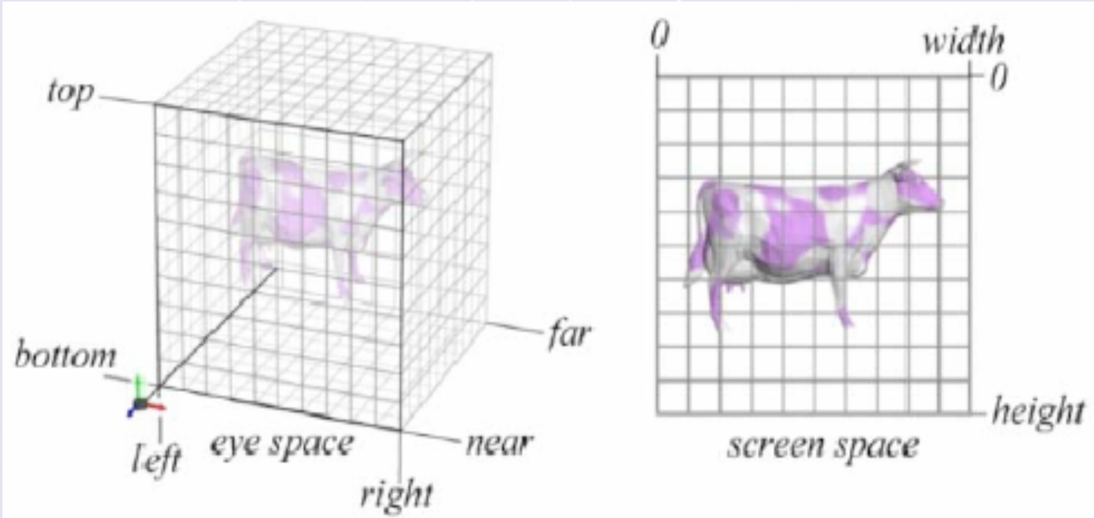
- Clipping:
  - Normalized Device Coordinates (NDC)
  - Remove parts that are out of the viewing volume



Graphic Pipeline

- Projection into screen space (2D)





Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

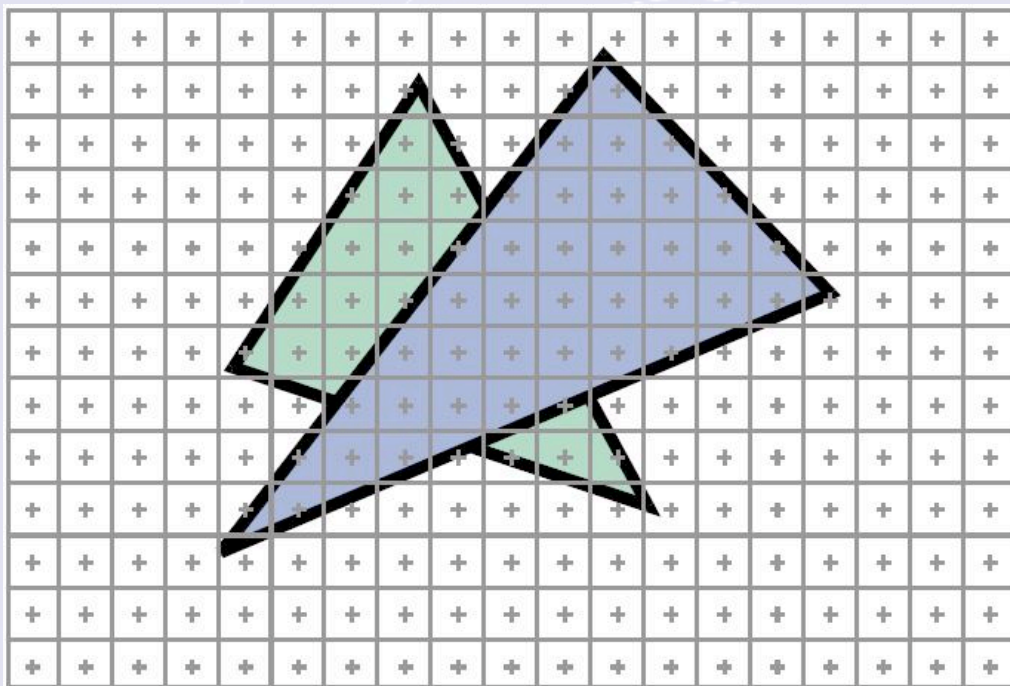
Projection (to Screen Space)

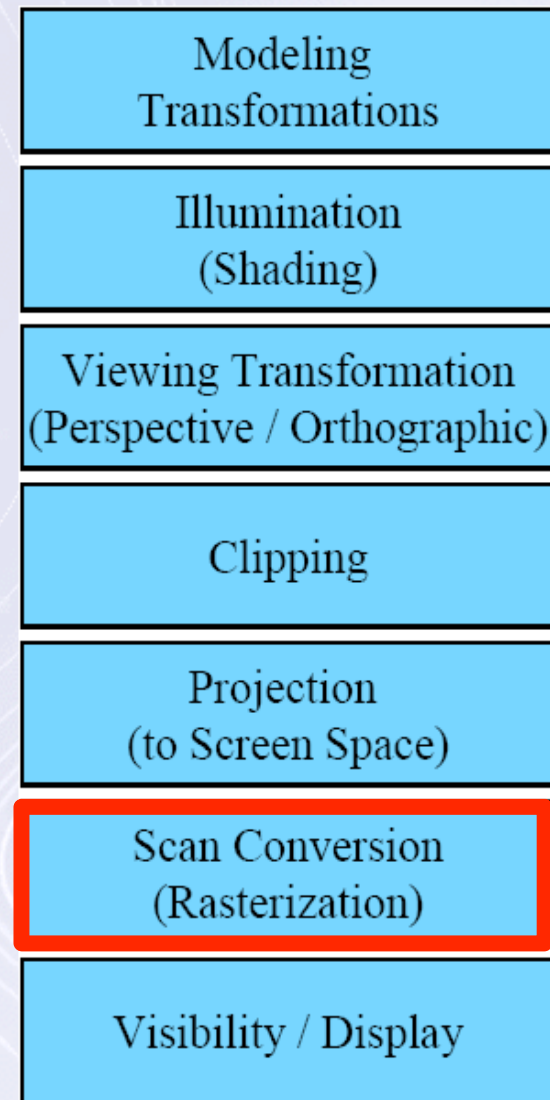
Scan Conversion (Rasterization)

Visibility / Display

Graphic Pipeline

- Rasterization :
  - Interpolates color and Z values given for vertices for each displayed fragment (pixel)





Graphic Pipeline

- Hidden surface removal (Z-buffer)
- Coordinate System Transformations are the most important to understand

Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

Graphic Pipeline

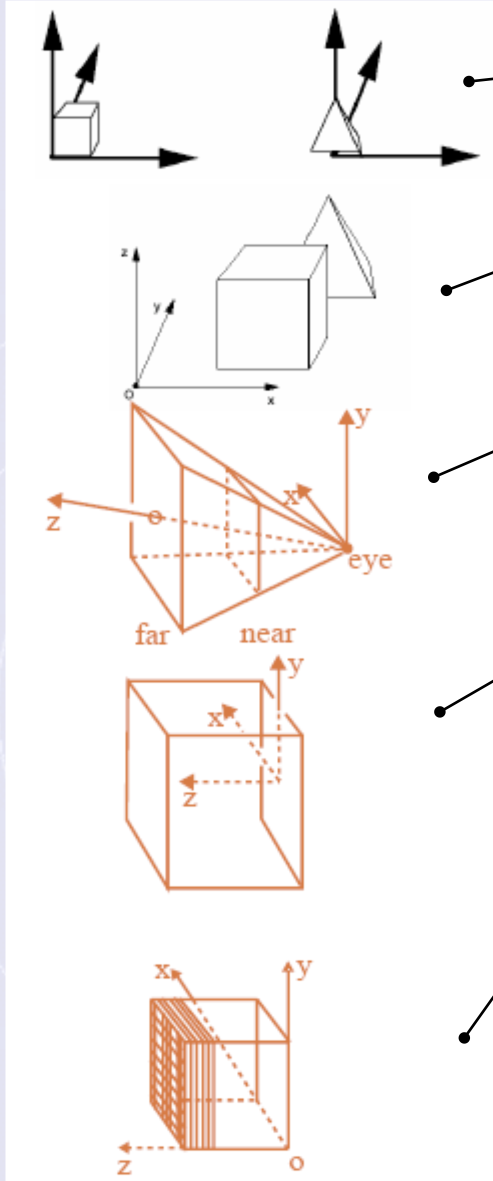
Object local CS

World CS

Camera local CS

Normalized Device CS

Screen CS



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

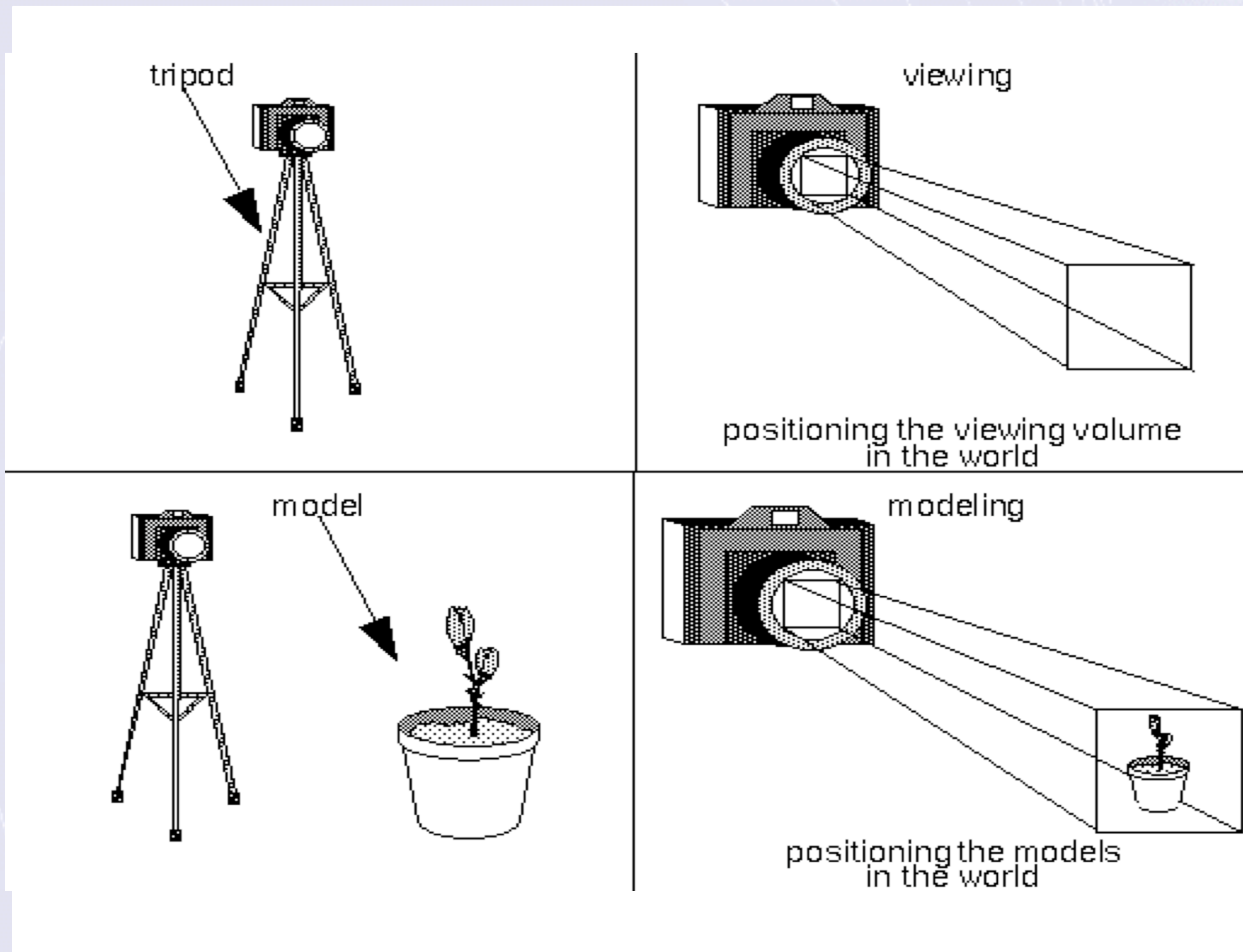
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

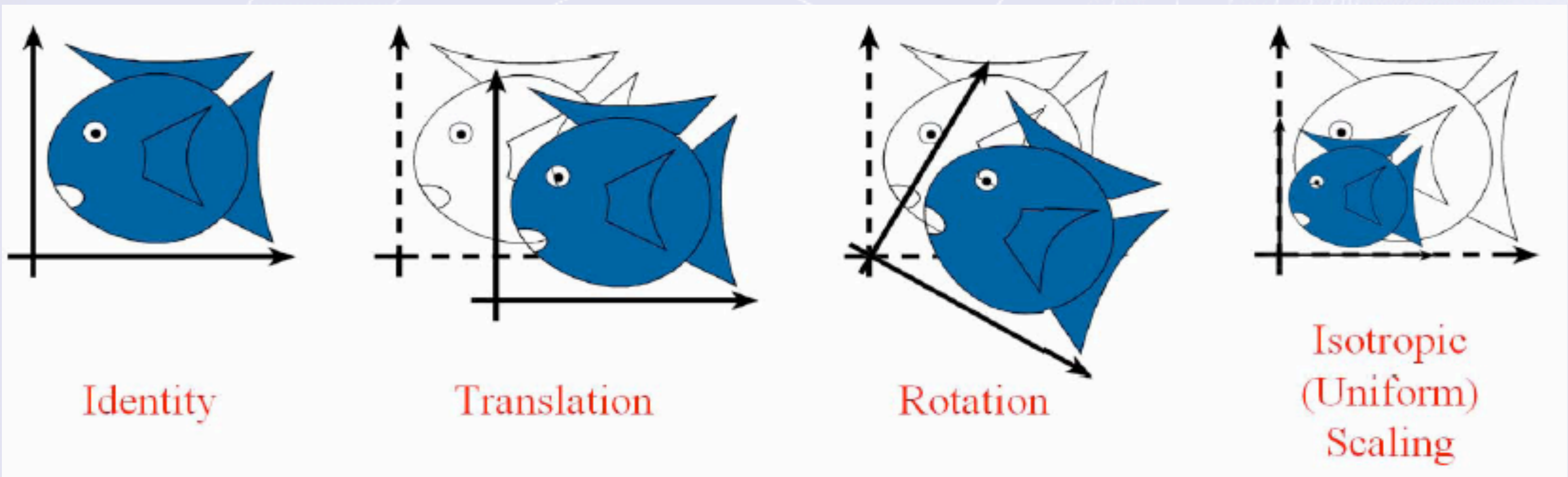
Projections and geometric transformations

- The Camera Metaphor



Geometric Transformations

- Affine Transformations:
  - Can be combined
  - Can be inverted
    - ☞ All but scale with coordinates to 0 !



Geometric Transformations

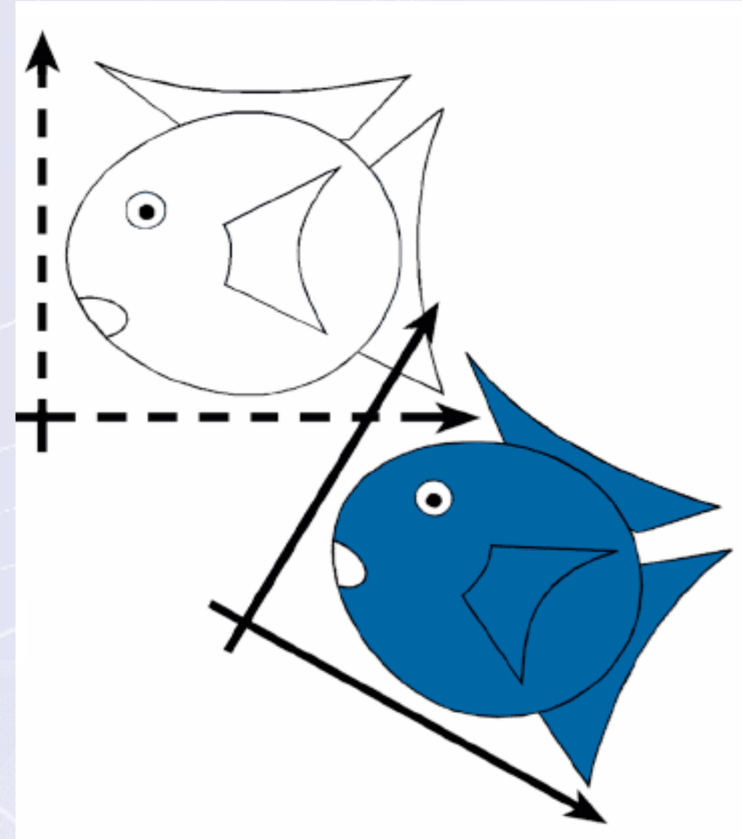
- Euclidean Transformations (rigid)
  - Preserving angles
  - Preserving distances

Identity

Translation

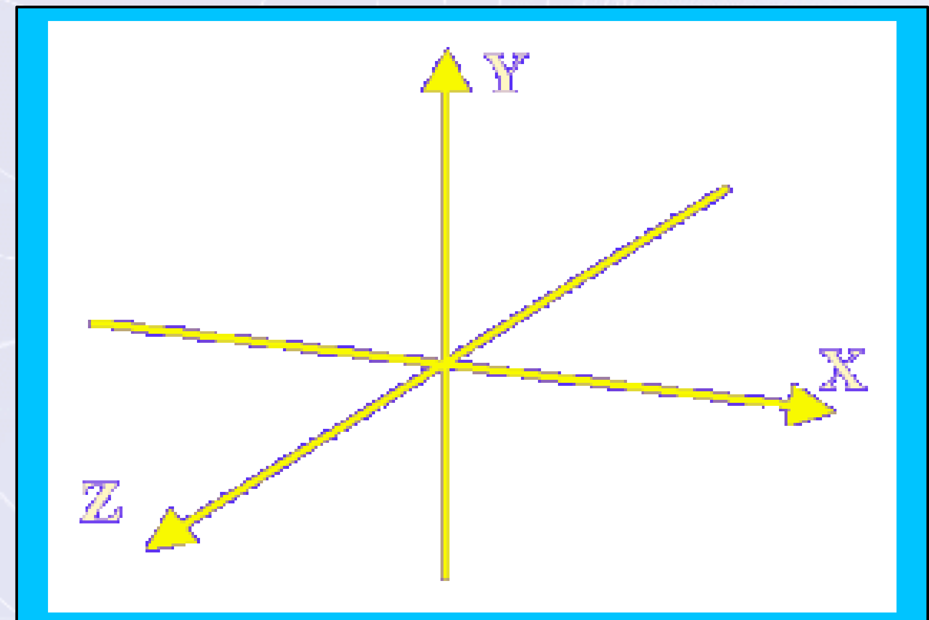
Rotation

Euclidean



Geometric Transformations

- Initial Coordinate system
  - Usually the XY plane is parallel to the screen
  - Y towards the top
  - X towards the right
  - Z goes out of the screen
  - It's a direct or right hand coordinate system



Geometric Transformations

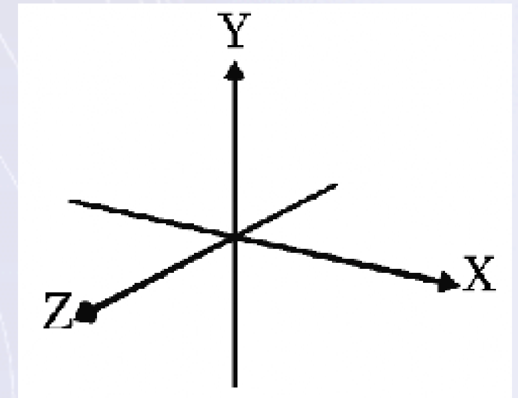
Translation

Geometric Transformations

Translation

- In order to move the object

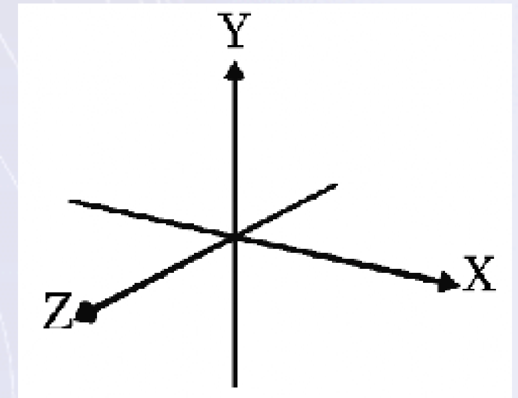
Geometric Transformations



Translation

- In order to move the object

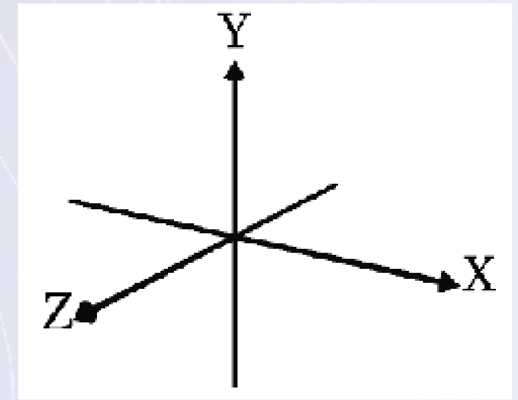
Geometric Transformations



Translation

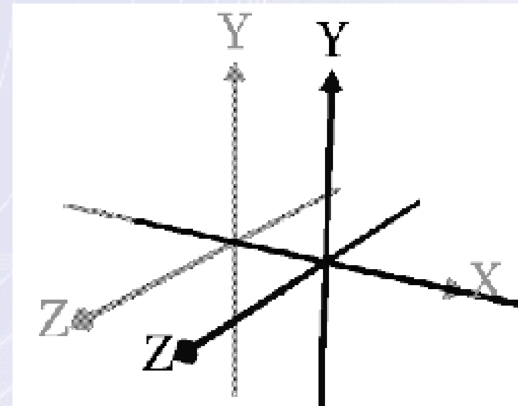
- In order to move the object
    - You translate the coordinate system (e.g. translation of 2 on X)

Geometric Transformations

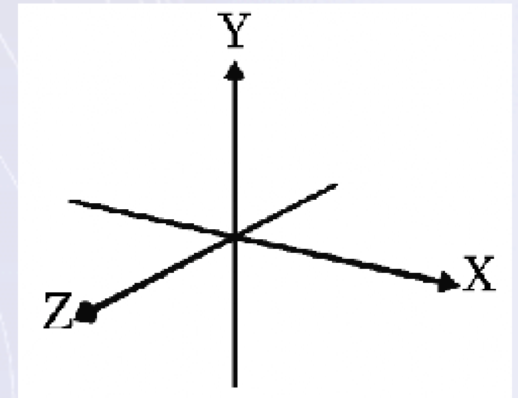


Translation

- In order to move the object
  - You translate the coordinate system (e.g. translation of 2 on X)

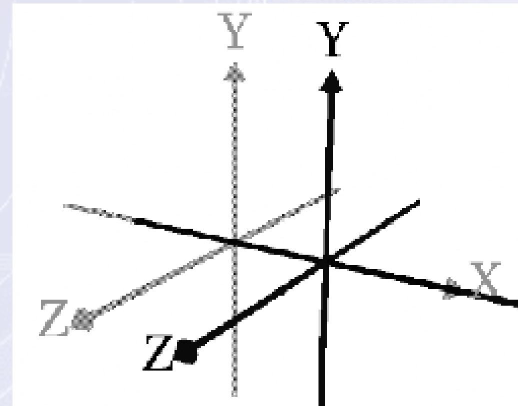


Geometric Transformations

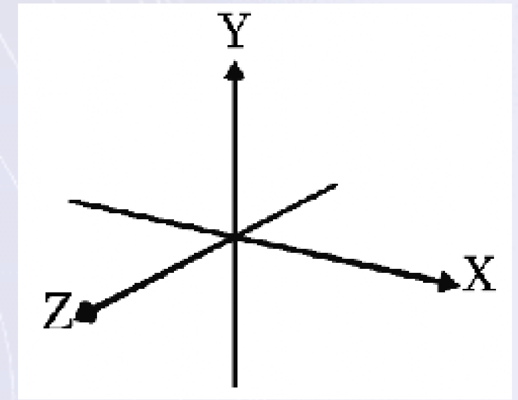


Translation

- In order to move the object
  - You translate the coordinate system (e.g. translation of 2 on X)
  - And then draw the object in the translated coordinate system

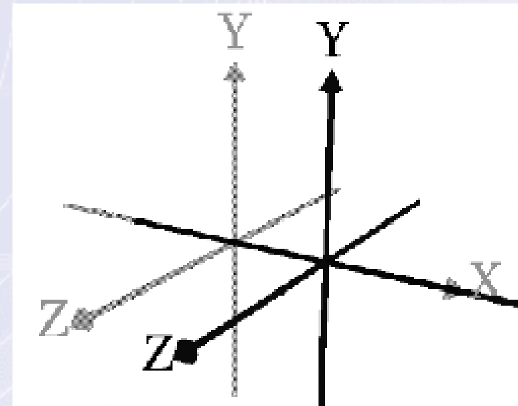


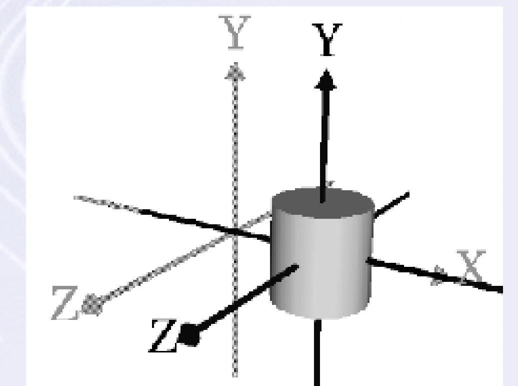
Geometric Transformations



Translation

- In order to move the object
  - You translate the coordinate system (e.g. translation of 2 on X)
  - And then draw the object in the translated coordinate system





Geometric Transformations

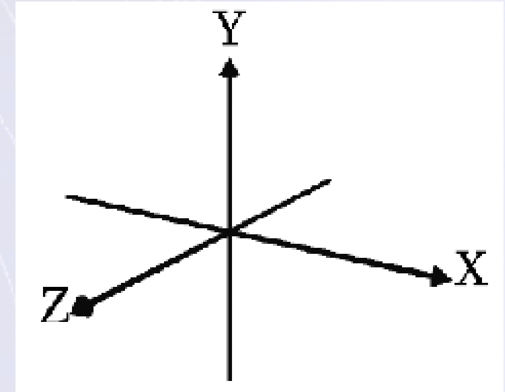
Rotation

Geometric Transformations

Rotation

- In order to rotate an object

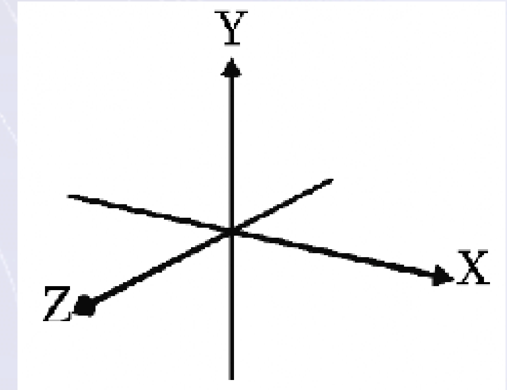
Geometric Transformations



Rotation

- In order to rotate an object

Geometric Transformations

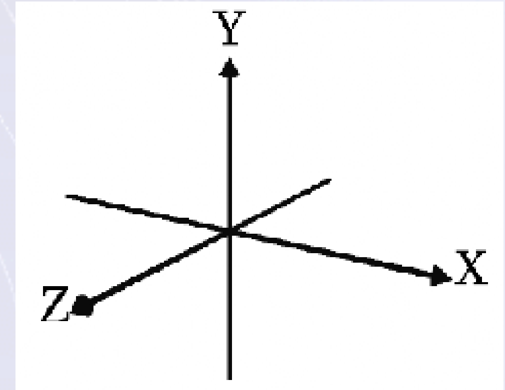


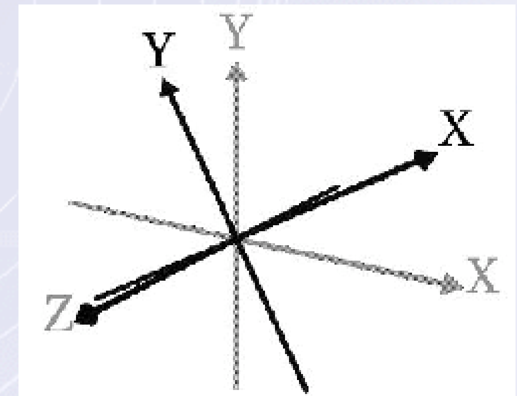
Rotation

- In order to rotate an object
  - You rotate the coordinate system (e.g. rotation of 40° around the Z axis)

Geometric Transformations

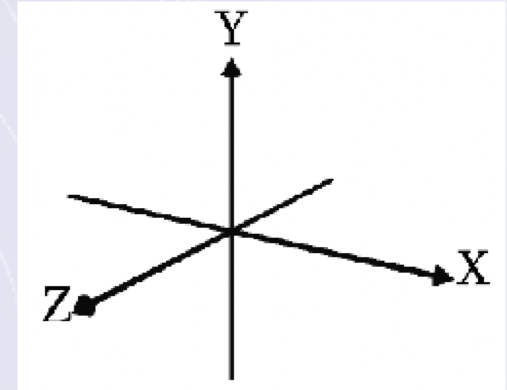
Rotation





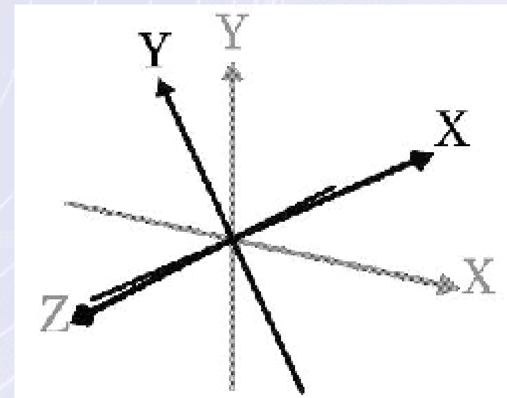
- In order to rotate an object
  - You rotate the coordinate system (e.g. rotation of 40° around the Z axis)

Geometric Transformations

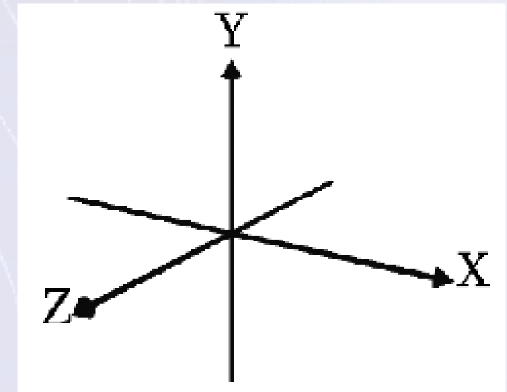


Rotation

- In order to rotate an object
  - You rotate the coordinate system (e.g. rotation of 40° around the Z axis)
  - You draw the object in the rotated coordinate system

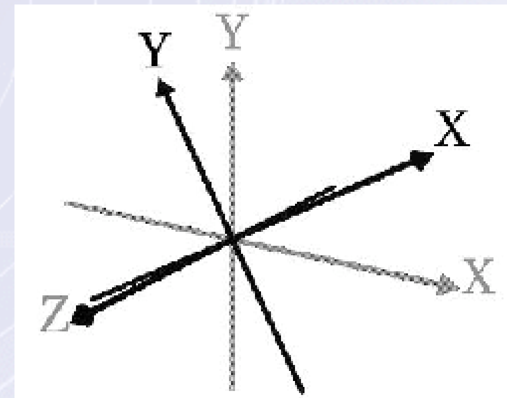


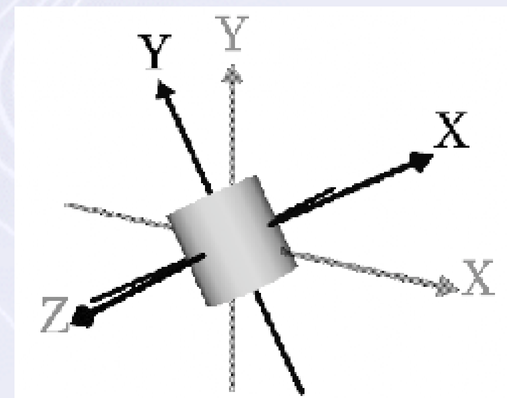
Geometric Transformations



Rotation

- In order to rotate an object
  - You rotate the coordinate system (e.g. rotation of 40° around the Z axis)
  - You draw the object in the rotated coordinate system

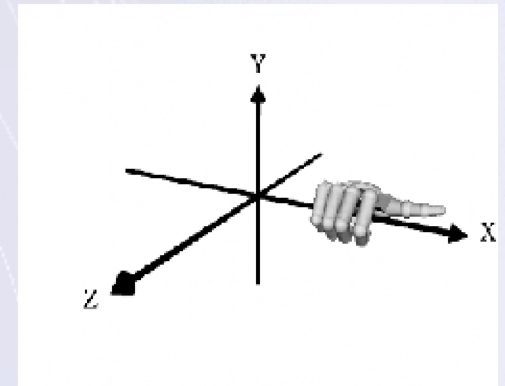


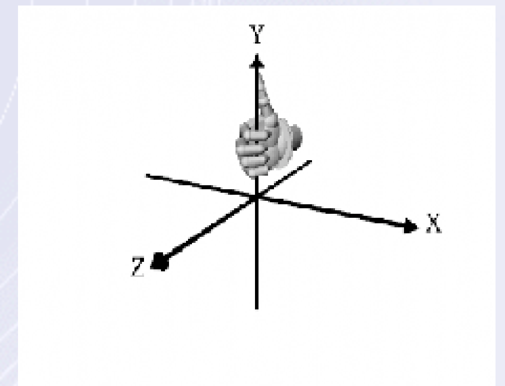


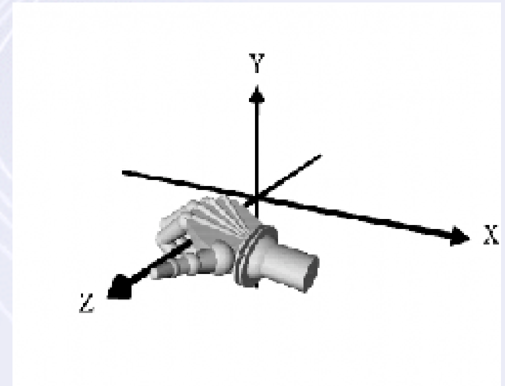
Geometric Transformations

Rotation

- Positive Rotations = trigonometric direction = counter clock wise
- The right hand rule:
  - If your thumb is in the axis direction
  - When you close your hand, your fingers are rotated positively

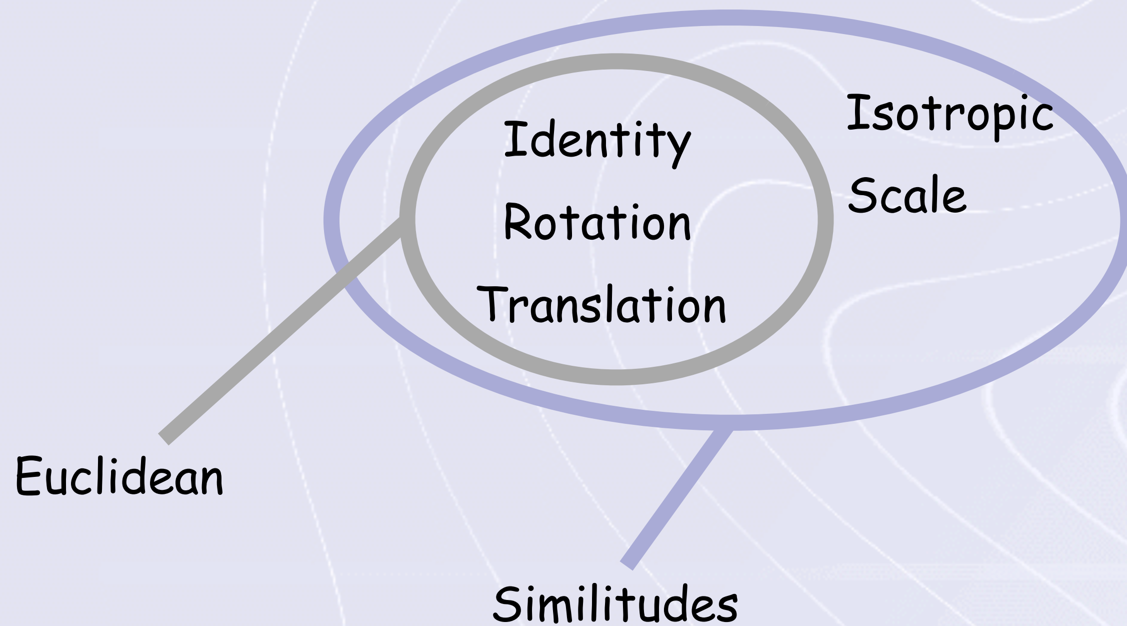


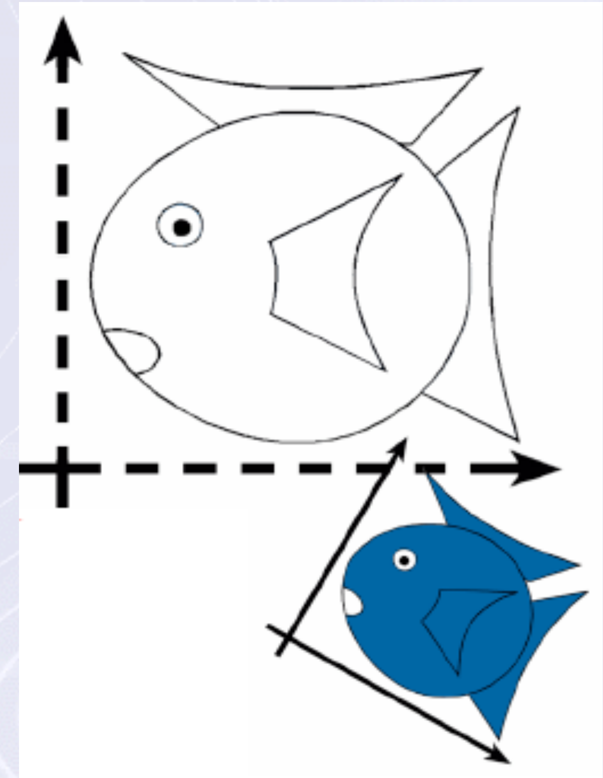




Geometric Transformations

- Similitudes
  - Preserving angles





Geometric Transformations

- Linear Transformations

Linear

Reflection, Scale, Shear

Isotropic

Scale

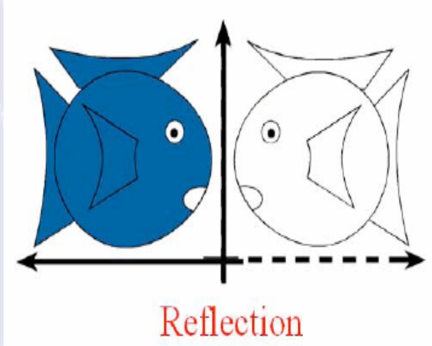
Identity

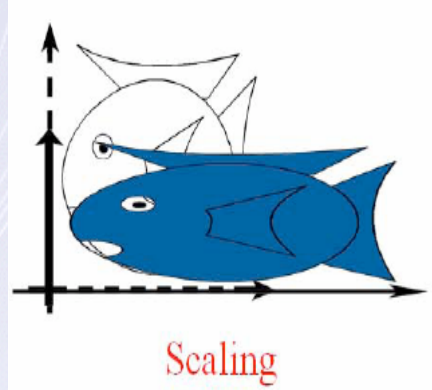
Rotation

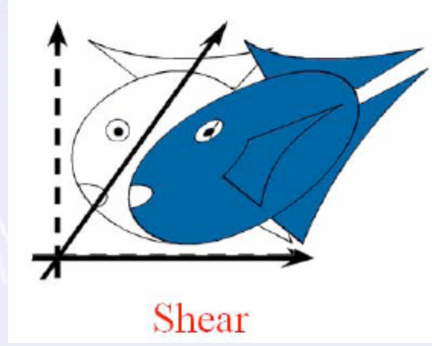
Translation

Euclidean

Similitudes


Reflection

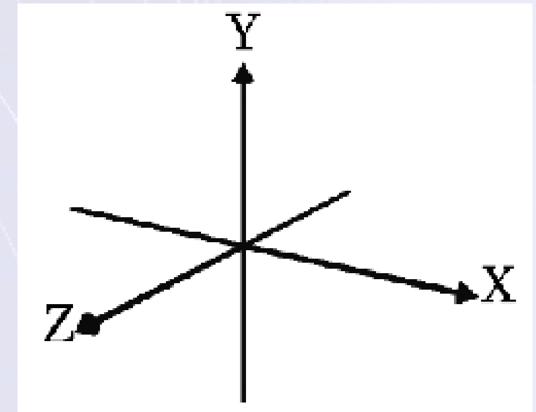

Scaling

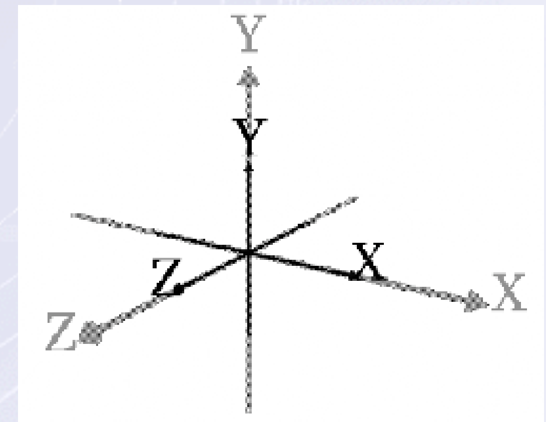

Shear

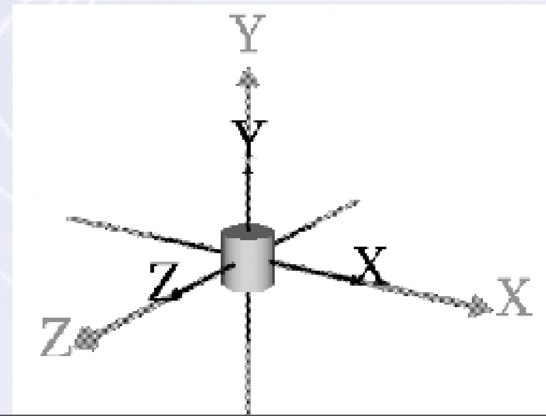
Geometric Transformations

Scaling

- To scale an object
  - You first scale the coordinate system (e.g. uniform scaling of 1/2)
  - You draw the object in the scaled coordinate system

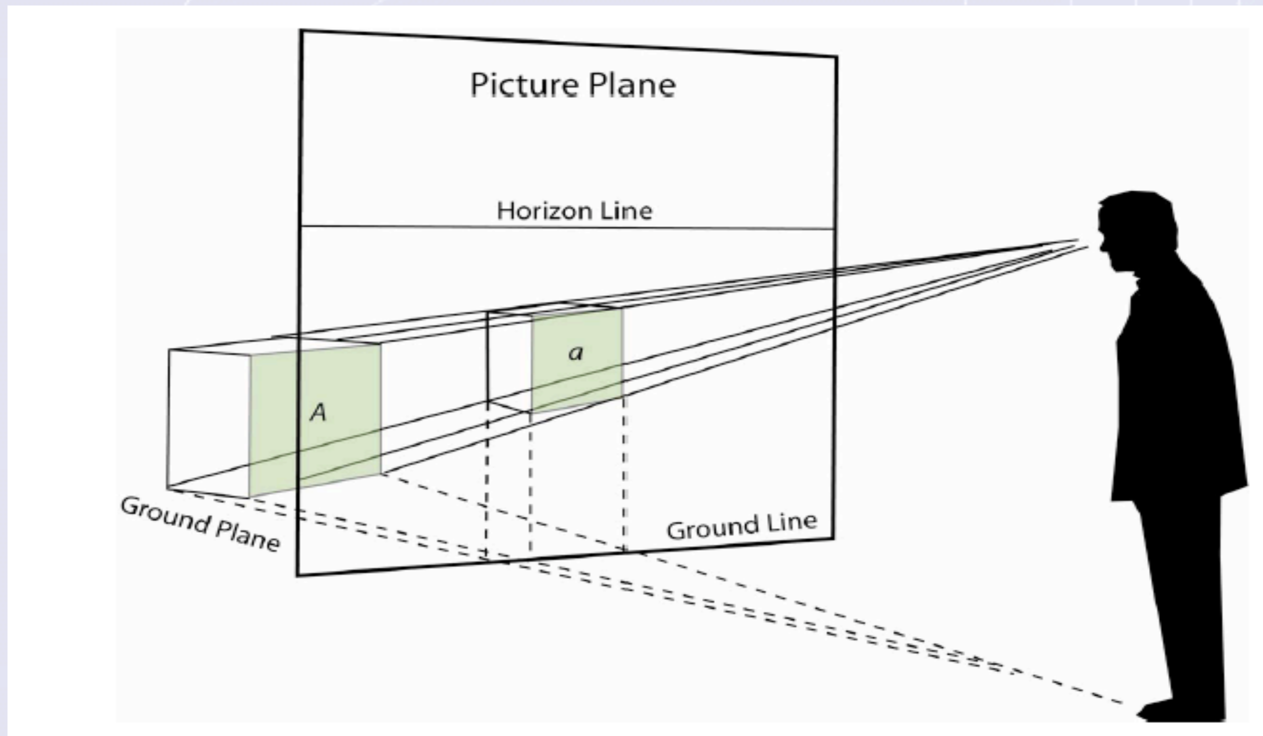






Geometric Transformations

- Projective Transformations:
  - Most generic case
  - **Cannot be inverted**
  - Do not conserve barycenter



Geometric Transformations

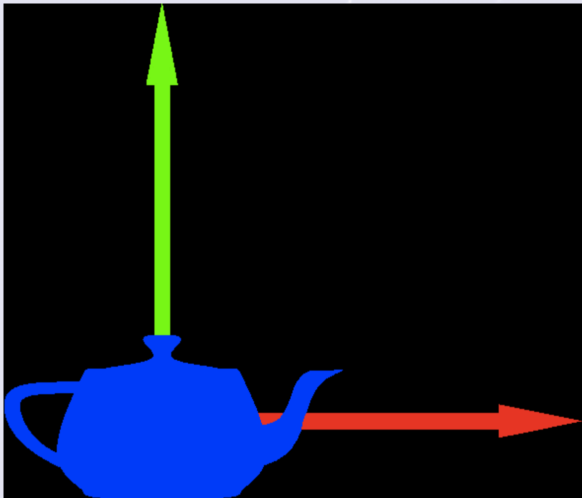
- Just a word on matrices
  - Matrices can represent any geometric transformation
- For this we use homogeneous coordinates
  - Point P = (x, y, z, w) ⇔ point (x/w, y/w, z/w, 1) in $\Re^2$ if $w \neq 0$
  - Vector V = (x, y, z, 0)
- Transformation Matrix (3D: 4x4)

$$\begin{array}{c} Current \\ Transformation \\ Matrix \end{array} = CTM = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

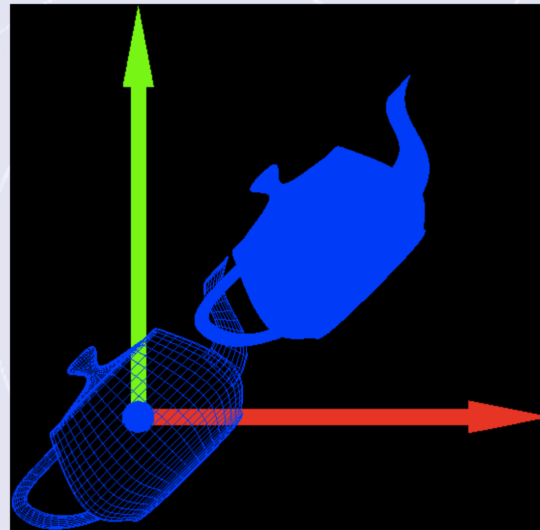
Geometric Transformations

- Chaining transformations: matrix product
  - Associative:

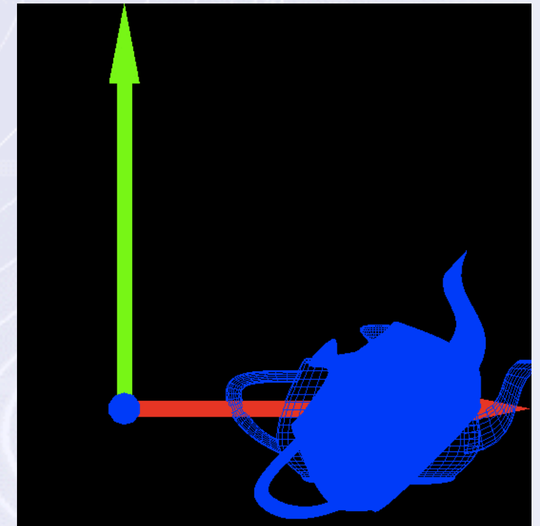
    p_0=(T0*(T1*T2))p =((T0*T1)*T2)p =(T0*T1*T2)p
  - But: **Not** commutative !



Initial Position



Rotation then Translation



Translation then Rotation

Projections

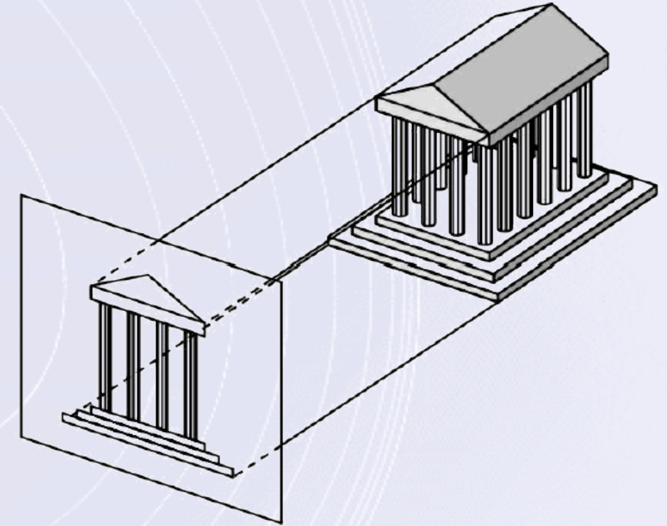
- Transformation world → image space
  - 3D world coordinates → 2D pixels
- Camera CS
  - Viewpoint: observer's position
  - View direction: observer's direction
  - Up direction: observer's vertical
- View volume
  - 2 projection types

Projections

- Transformation world → image space
  - 3D world coordinates → 2D pixels
- Camera CS
  - Viewpoint: observer's position
  - View direction: observer's direction
  - Up direction: observer's vertical
- View volume
  - 2 projection types
    - Parallel (Orthographic)

Projections

- Transformation world → image space
  - 3D world coordinates → 2D pixels
- Camera CS
  - Viewpoint: observer's position
  - View direction: observer's direction
  - Up direction: observer's vertical
- View volume
  - 2 projection types
    - Parallel (Orthographic)



Projections

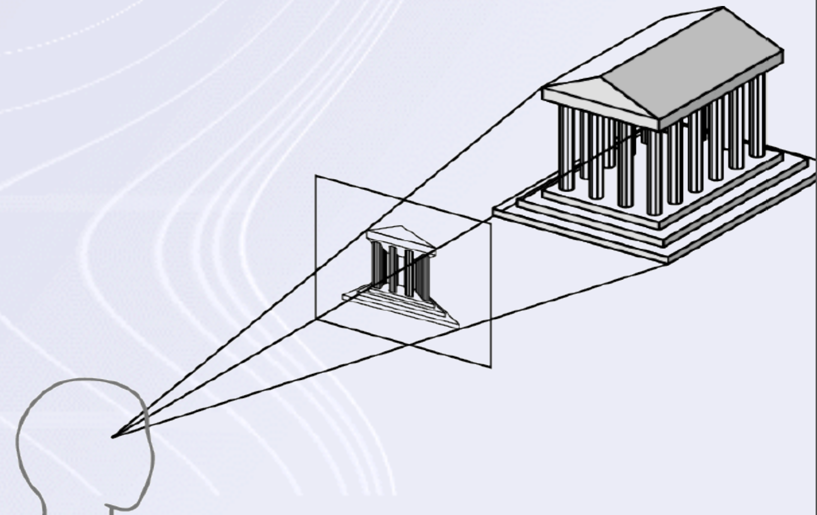
- Transformation world → image space
  - 3D world coordinates → 2D pixels
- Camera CS
  - Viewpoint: observer's position
  - View direction: observer's direction
  - Up direction: observer's vertical
- View volume
  - 2 projection types
    - Parallel (Orthographic)

Projections

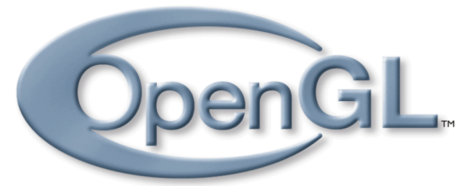
- Transformation world → image space
  - 3D world coordinates → 2D pixels
- Camera CS
  - Viewpoint: observer's position
  - View direction: observer's direction
  - Up direction: observer's vertical
- View volume
  - 2 projection types
    - Parallel (Orthographic)
    - Perspective

Projections

- Transformation world → image space
  - 3D world coordinates → 2D pixels
- Camera CS
  - Viewpoint: observer's position
  - View direction: observer's direction
  - Up direction: observer's vertical
- View volume
  - 2 projection types
    - Parallel (Orthographic)
    - Perspective



Introduction to

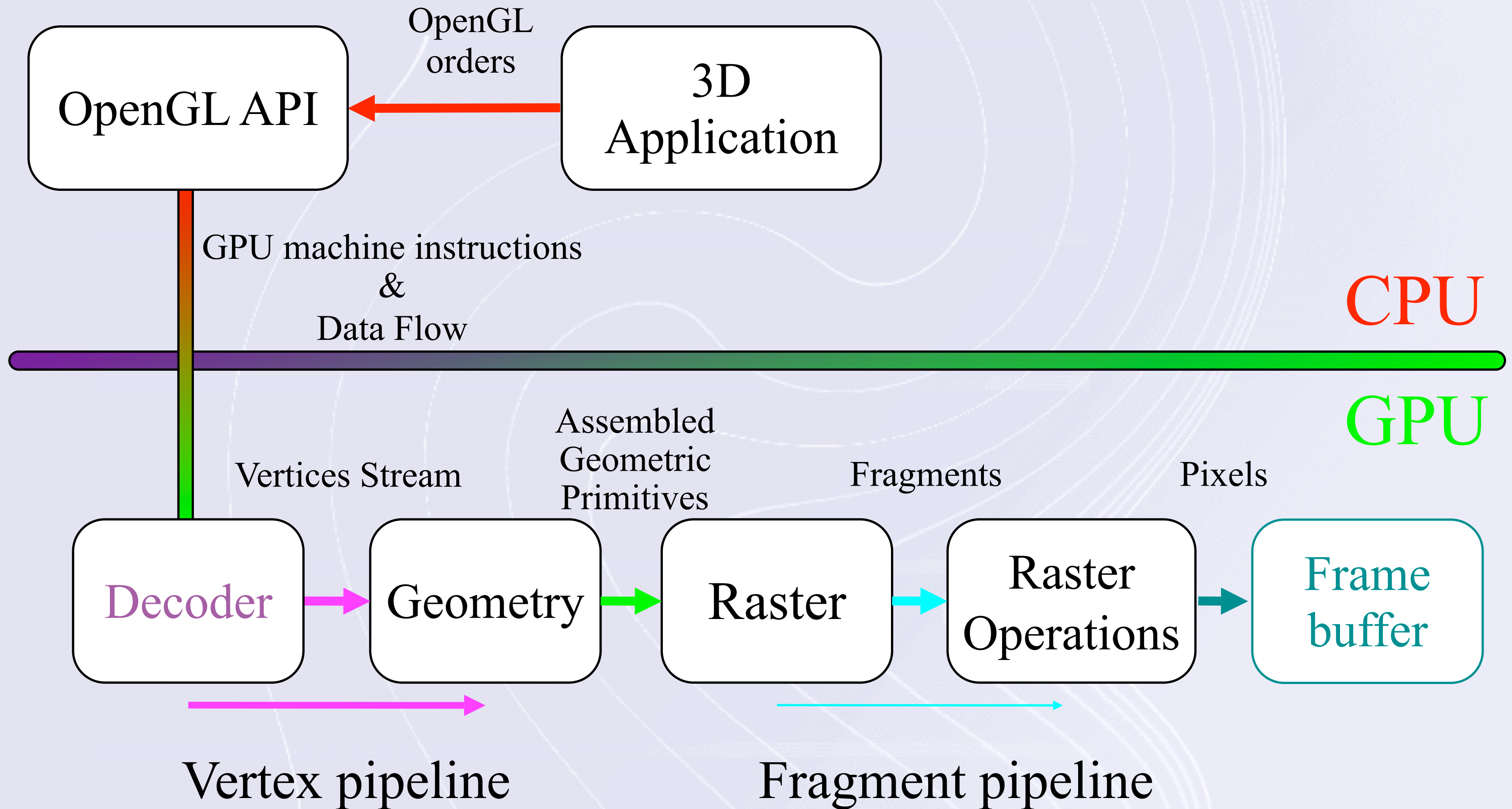


- OpenGL : Open Graphics Library
  - CG Standard (Architecture Review Board: ARB)
  - API for the graphics hardware
    - about 200 functions/orders used to create 3D animated applications
  - States
    - State values: viewing volume data, drawing properties (color, point/lines width, etc.), materials, light properties, etc.

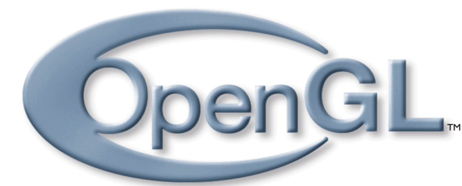
- OpenGL is not
  - A GUI system (use Windows API, X-Window, Swing, AWT...)
  - A geometric modeler (use 3D Studio, Maya...)

Introduction to OpenGL

- Graphic Pipeline



Basics of



- General Structure of an OpenGL function (C)

```
void gl…{2,3,4}{s,i,f,d}[v] (TYPE coords);
```

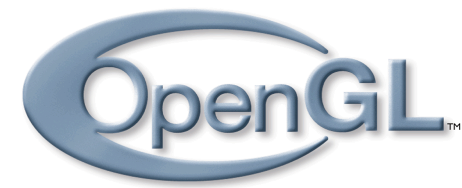
2 (x, y)
3 (x, y, z)
4 (x, y, z, w)

v vector

s 16 bits int [short]
i 32 bits int [int]
f 32 bits fp [float]
d 64 bits fp [double]

TYPES --
GLshort -- s
GLfloat -- f
GLdouble -- d
GLint -- i
GLxx * -- v

Basics of



- Vertices: geometric primitives base

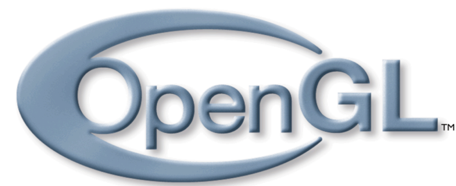
  glVertex {2,3}{s,i,f,d}[v]()

  - Given between glBegin() and glEnd()
  - Select a drawing mode

```
glBegin(mode);
    glVertex*(coordinates);
    glVertex*(coordinates);
    .
    .
    glVertex*(coordinates);
glEnd();
```

MODES
GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_POLYGON
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP

Basics of

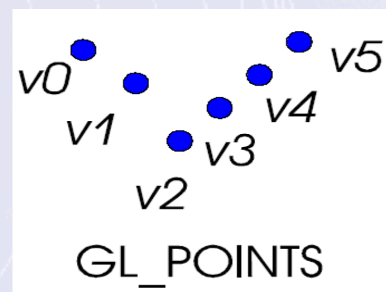


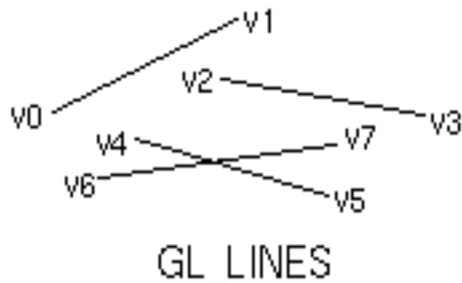
- glVertex* (coordinates)
  - Examples

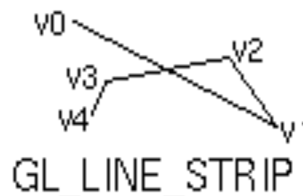
```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.1415926535898);
glVertex3f( 2.3f, 12.0f, -4.8f);

GLdouble dvect[3]={5.0, 9.0, 1435.0};
glVertex3dv(dvect);
```

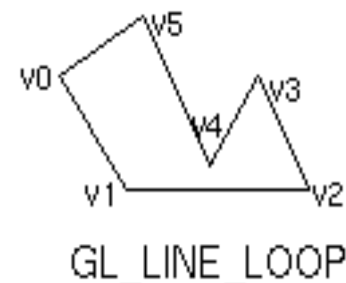
Basics of OpenGL

- Drawing Modes
  - Points
  - Line segments
    - independent
    - poly-lines
    - hollow polygons



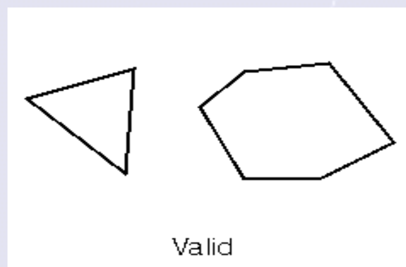


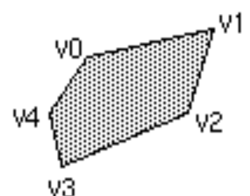


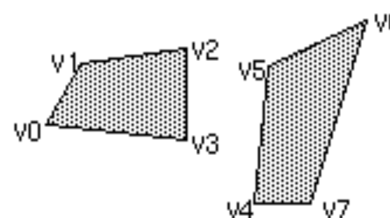


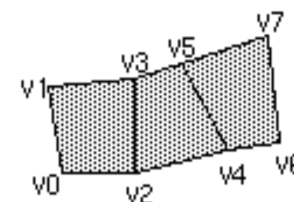
Basics of OpenGL

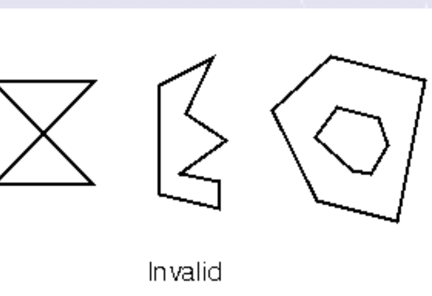
- Drawing Modes
  - Polygonal
    - independent polygons, triangles and quads (only convex!)
    - triangle fans, triangle strips, quad strips

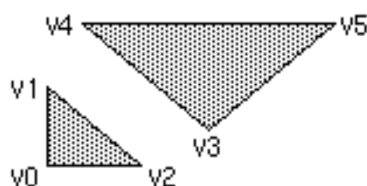


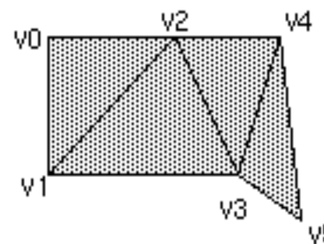


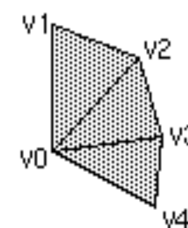




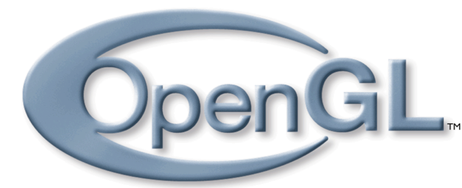








Basics of



- Polygon Orientation (front facing)
  - Convention: vertex order using the trigonometric direction
  - If you need to change that:

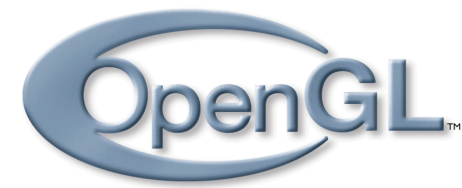
```
glFrontFace(mode)
```

    - Mode: GL_CCW (counter clock wise - default), GL_CW (clock wise)

- Polygon Drawing modes

```
glPolygonMode(face, mode)
```

    - Side: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
    - Mode: GL_POINT, GL_LINE, GL_FILL

Introduction to 

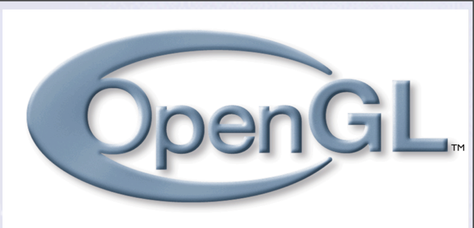
- APIs for UNIX

Application

OpenGL | Glx | X-Window

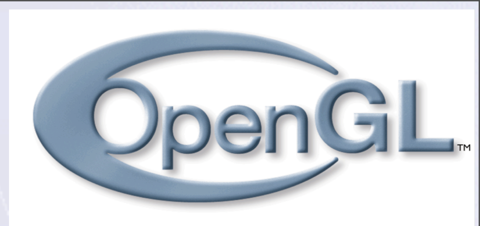
Driver

Hardware

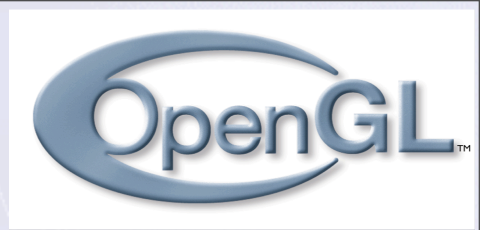
Introduction to 

- APIs for UNIX

Introduction to

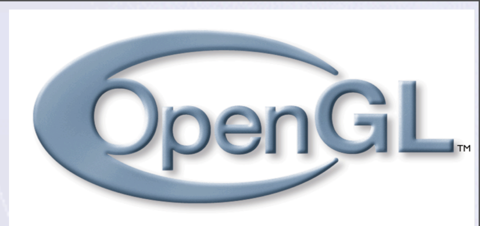


- APIs for UNIX

Introduction to 

- APIs for Windows

Introduction to



- APIs for Java (JOGL)

Application

GLU

OpenGL

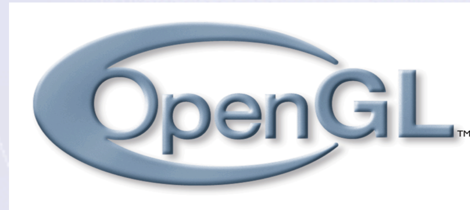
Swing

AWT

Driver

Hardware

Projections and transformations

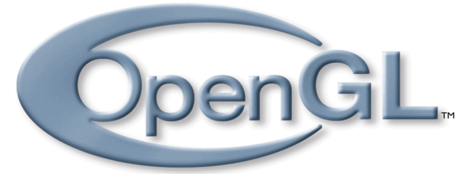


- 2 Matrix modes
  - void glMatrixMode(Glenum mode)
  - Mode: GL_PROJECTION
    - View Volume
    - Projection
      - Orthographic
      - Perspective
  - Mode: GL_MODELVIEW
    - Transformations for modeling and for viewing (inverted)
      - Scale
      - Translation
      - Rotation

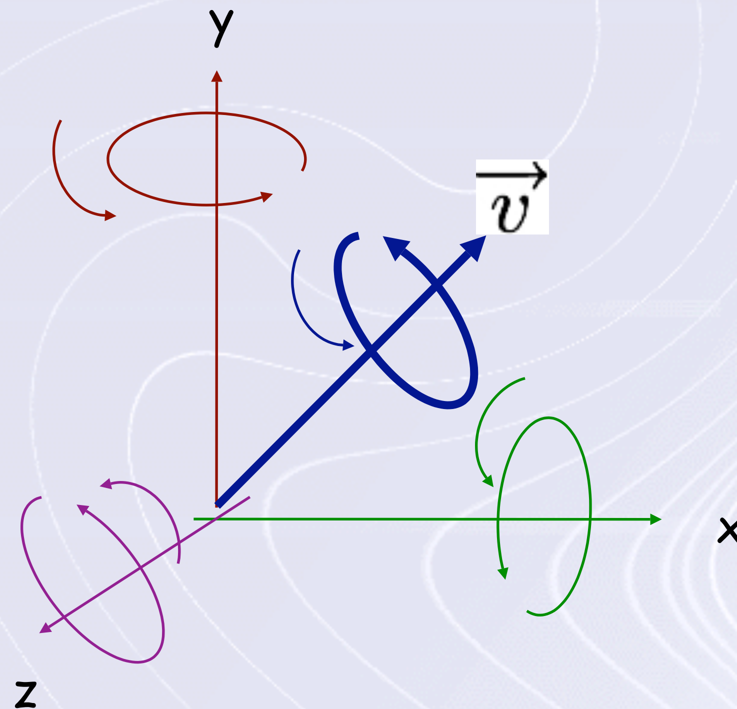
glScale{fd} (TYPE x, TYPE y, TYPE z);

glTranslate{fd} (TYPE x, TYPE y, TYPE z);

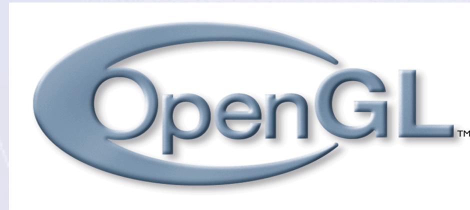
Geometric Transformations



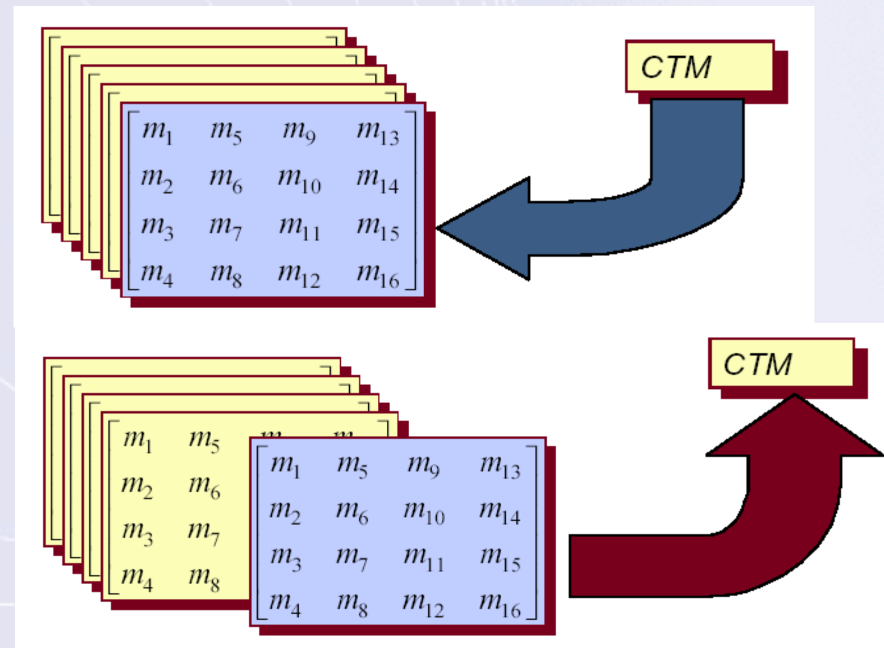
- Rotation
  - glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z);



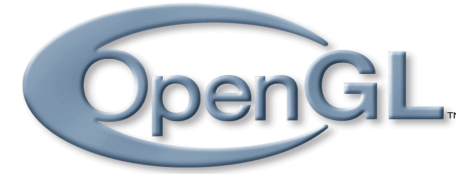
Geometric Transformations



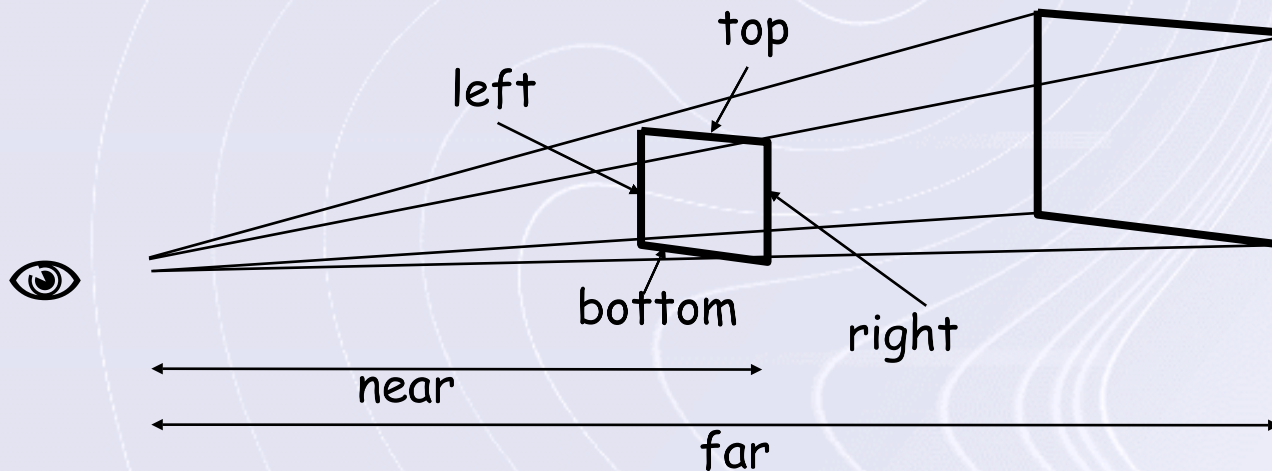
- Matrix Stack Management
  - void glLoadIdentity(void)
  - void glPushMatrix(void)
  - void glPopMatrix(void)
  - glLoadMatrix{fd}(TYPE * mat)
    - CTM = mat
  - glMultMatrix{fd}(TYPE *mat)
    - CTM = CTM x mat
  - void glGet{Float, Double}v (Glenum pname, glFloat *mat)
    - m = CTM
    - pname = GL_MODELVIEW_MATRIX or GL_PROJECTION_MATRIX



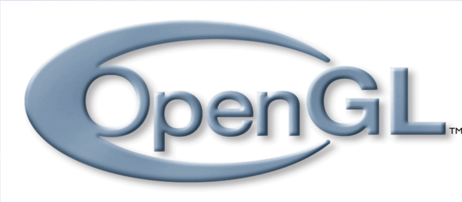
Projections and Viewing Volumes



- Perspective Projection
  - void glFrustum(GLdouble left, GLdouble right, Gldouble bottom, GLdouble top, GLdouble near, GLdouble far)



Colors



RGBA or Color Index

color index mode

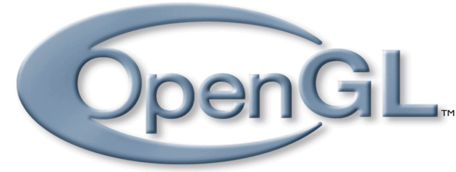
	Red	Green	Blue
0			
1			
2			
3			
◆◆◆			
24	123	219	74
25			
26			
◆◆◆			

Frame

Display

Direct or RGBA mode

Colors



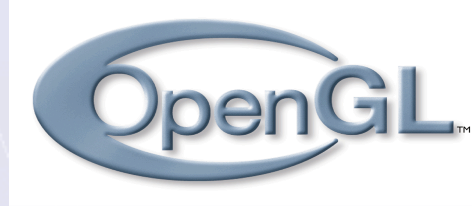
- Direct mode (RGB{A})

```
void glColor3{b,s,i,f,d,ub,us,ui}(TYPE r,TYPE g,TYPE b);
void glColor4{b,s,i,f,d,ub,us,ui} (TYPE r, TYPE g, TYPE b,TYPE a );
void glColor{3 4}{b,s,i,f,d,ub,us,ui}v (TYPE *v );
```

r red
g green
b blue
a alpha
v vector of 3 or 4 values

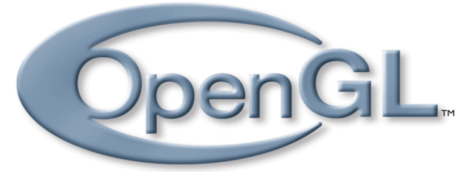
b -- byte
s -- short
i -- int
f -- float
d -- double
ub-- unsigned byte
us-- unsigned short
ui-- unsigned int

State



- All rendering attributes are encapsulated in the OpenGL State
- rendering styles
- shading
- lighting
- texture mapping

State



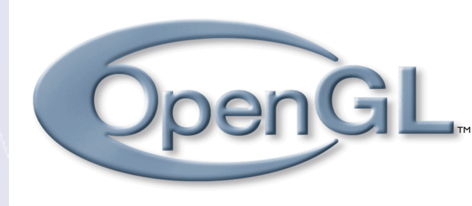
Appearance is controlled by the current state

```
for each ( primitive to render ) {
update OpenGL state
render primitive
}
```

Manipulating vertex attributes is the most common way to manipulate the state

- glColor*() / glIndex*()
- glNormal*()
- glTexCoord*()

State



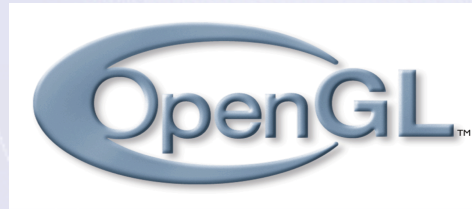
Setting State

- glPointSize(size);
- glLineStipple(repeat, pattern);
- glShadeModel(GL_SMOOTH);

Enabling Features

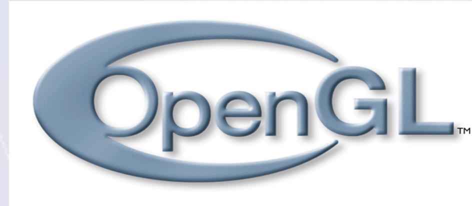
- glEnable(GL_LIGHTING);
- glDisable(GL_TEXTURE_2D);
- glEnable(GL_DEPTH_TEST);

GLCanvas and GLEventListener



- GLCanvas: AWT Canvas for GL drawing
- Other solution: GLJPanel, a Swing panel for GL drawing (a bit slower)
  - Both implement the GLAutoDrawable interface
  - Main method: GL getGL(); All GL orders are methods of GL.
- But where do we put draw orders ?
  - GLEventListener is an interface you must implement for this
  - void init(GLAutoDrawable d) is where you do your initializations
  - void reshape(GLAutoDrawable d) is called when the window dimensions change
  - void display(GLAutoDrawable d) is where you do the per-frame drawing

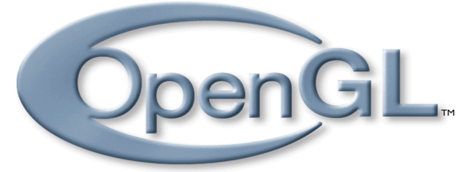
GLCanvas and GLEventListener



- First Basic Example:

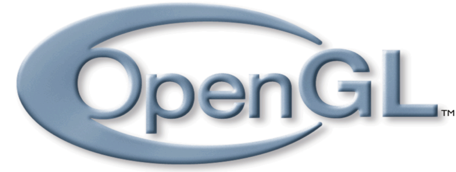
```
import javax.media.opengl.*;
import java.awt.Frame;

public class Basic implements GLEventListener {
    public void display(GLAutoDrawable drawable) {
      GL gl = drawable.getGL();
      gl.glBegin(GL.GL_POLYGON); // draw a square
          gl.glVertex2f(-0.5f, -0.5f);
          gl.glVertex2f(-0.5f, 0.5f);
          gl.glVertex2f(0.5f, 0.5f);
          gl.glVertex2f(0.5f, -0.5f);
      gl.glEnd();
    }
    public static void main(String[ ] args) {
      Frame frame = new Frame("Basic"); // create the frame
      frame.setSize(500, 500); // give it a size
      GLCanvas canvas = new GLCanvas(); // create the canvas
      Basic basic = new Basic(); // create the listener
      canvas.addGLEventListener(basic); // add listener to canvas
      frame.add(canvas);//add canvas to window
      frame.setVisible(true); // show window
    }
```

3D Rendering



- In order to do some real 3D rendering we need to:
  - Activate hidden surface removal (in init)
  - Specify a frustum when the window change (in reshape)
  - Clear the image buffer and the depth buffer (in display)
  - Transform the initial CS (in display)
  - Choose a color (in display)
  - Draw some primitives (in display)
  - ...

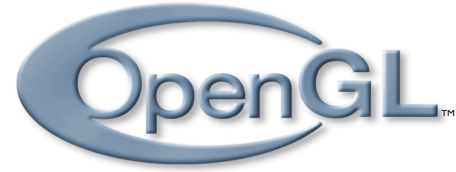
3D Rendering



- Another (more complex) example

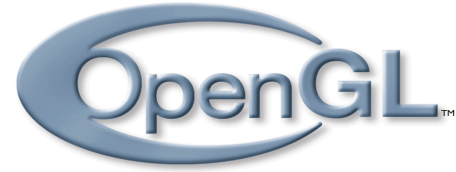
```
import javax.media.opengl.*;
import java.awt.Frame;

public class Complex implements GLEventListener {
     public static void main(String[ ] args) {
        Frame frame = new Frame("Basic"); // create the frame
        frame.setSize(500, 500); // give it a size
        GLCanvas canvas = new GLCanvas(); // create the canvas
        Complex basic = new Complex(); // create the listener
        canvas.addGLEventListener(basic); // add listener to canvas
        frame.add(canvas);//add canvas to window
        frame.setVisible(true); // show window
      }
    public void init(GLAutoDrawable drawable) {
        GL gl = drawable.getGL();
        gl.glEnable(GL.GL_CULL_FACE);
        gl.glEnable(GL.GL_DEPTH_TEST);
    }
...
```

3D Rendering



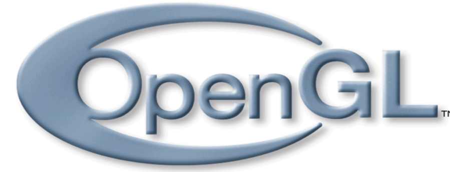

```
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
      GL gl = drawable.getGL();
      float ratio = (float)height / (float)width; // compute height/width ratio
      gl.glMatrixMode(GL.GL_PROJECTION);
      gl.glLoadIdentity();
      gl.glFrustum(-1.0f, 1.0f, -ratio, ratio, 5.0f, 6000.0f);
      gl.glMatrixMode(GL.GL_MODELVIEW);
}
public void display(GLAutoDrawable drawable) {
      GL gl = drawable.getGL();
      gl.glClearColor(0.3f, 0.3f, 0.99f, 1.0f); // a blue sky
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
      // set the view transformation
      gl.glLoadIdentity();
      gl.glTranslatef(0.f, -4.f, 0.0f); // look from a bit above the floor
      // draw the floor
      drawFloor(gl);
      // draw some pyramids
      drawScenery(gl);
 }
```

3D Rendering



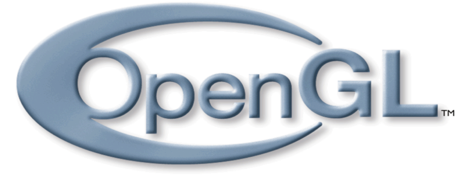
```
// draw the floor
private void drawFloor(GL gl)
{
    gl.glBegin(GL.GL_QUADS);
        gl.glColor3f(0.3f, 0.7f, 0.3f);
        gl.glVertex3f(-1000.0f, 0.0f,  1000.0f);
        gl.glVertex3f( 1000.0f, 0.0f,  1000.0f);
        gl.glVertex3f( 1000.0f, 0.0f, -1000.0f);
        gl.glVertex3f(-1000.0f, 0.0f, -1000.0f);
    gl.glEnd();
 }
```

3D Rendering




```
 // draw some pyramids
private void drawScenery(GL gl)
{
  for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
      gl.glPushMatrix();
      gl.glTranslatef(i*200.f - 900.f, 5.f, j*200.f - 1000.f);
      gl.glScalef(10.f, 10.f, 10.f);
      gl.glColor3f(((float)i) / 10.0f, ((float)j) / 10.0f, ((float)i+j) / 20.0f);
      gl.glBegin(GL.GL_TRIANGLES);
        gl.glVertex3f(0.0f, 1.0f, 0.0f);     //Top Of Triangle (Front)
        gl.glVertex3f(-1.0f, -1.0f, 1.0f);   //Left Of Triangle (Front)
        gl.glVertex3f(1.0f, -1.0f, 1.0f);    //Right Of Triangle (Front)

        gl.glVertex3f(0.0f, 1.0f, 0.0f);     //Top Of Triangle (Right)
        gl.glVertex3f(1.0f, -1.0f, 1.0f);    //Left Of Triangle (Right)
        gl.glVertex3f(1.0f, -1.0f, -1.0f);   //Right Of Triangle (Right)

        gl.glVertex3f(0.0f, 1.0f, 0.0f);     //Top Of Triangle (Back)
        gl.glVertex3f(1.0f, -1.0f, -1.0f);   //Left Of Triangle (Back)
        gl.glVertex3f(-1.0f, -1.0f, -1.0f);  //Right Of Triangle (Back)

        gl.glVertex3f(0.0f, 1.0f, 0.0f);     //Top Of Triangle (Left)
        gl.glVertex3f(-1.0f, -1.0f, -1.0f);  //Left Of Triangle (Left)
        gl.glVertex3f(-1.0f, -1.0f, 1.0f);   //Right Of Triangle (Left)
      gl.glEnd();
      gl.glPopMatrix();
    }
  }
}
}
```

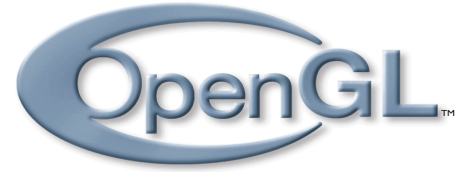
Animations



- If we want the scene to be re-drawn automatically, we need an Animator set on the Canvas.
  - It will create a thread which will call the display method of the canvas (which then calls your display method)
- Example (changing a bit the previous example):
  - Add an attribute: `float angle = 0;`
  - In the main:

```
...
frame.add(canvas);//add canvas to window
Animator animator = new Animator(canvas); // add the animator
animator.start(); // start the animator thread
frame.setVisible(true); // show window
```

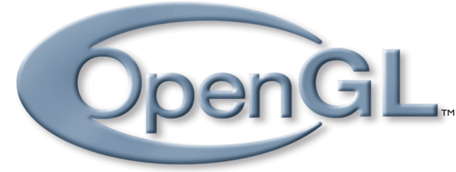
Animations



- Example (changing a bit the previous example):
  - In the init method: add `gl.glEnable(GL.GL_DOUBLEBUFFER);`
  - In the display method:

```
...
// set the view transformation
gl.glLoadIdentity();
gl.glTranslatef(0.f, -4.f, 0.0f); // look from a bit above the floor
// rotate the world
gl.glRotatef(angle, 0.f, 1.f, 0.f);
angle+=0.05f;
// draw the floor
drawFloor(gl);
 ...
```

Conclusion



- The only limit is your imagination
- You'll find jogl here:

  https://jogl.dev.java.net

- Setting up eclipse for jogl
  - Create a new project
  - Add the jogl.jar and gluegen-rt.jar to its library path
  - Set the native library path on jogl.jar to the place where you have the native libraries (right-click on jogl.jar, Properties, Native Libraries)
  - Code...