ROYAL INSTITUTE OF TECHNOLOGY

DH2323 DGI16

INTRODUCTION TO COMPUTER GRAPHICS AND INTERACTION

RASTERISED RENDERING Part II

Christopher Peters

HPCViz, KTH Royal Institute of Technology, Sweden

chpeters@kth.se

http://kth.academia.edu/ChristopherEdwardPeters

Based on DGI12 notes by Carl Henrik Ek

Rasterisation

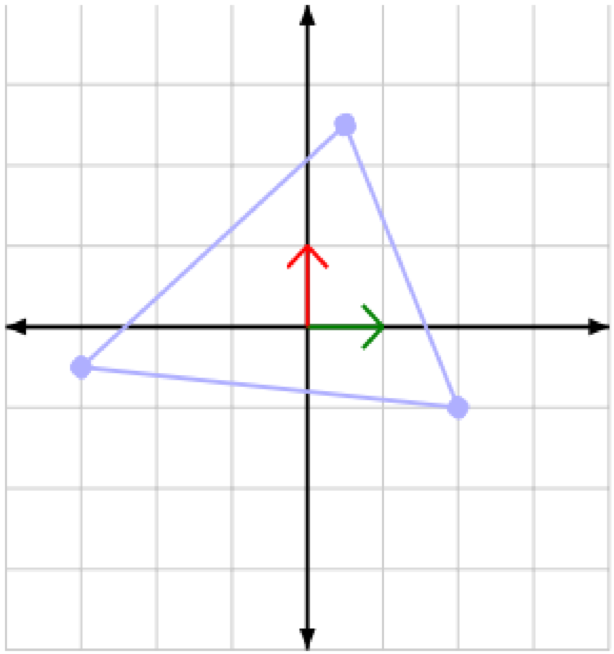
- Scanline: object order based
- *Fragments*
  - Data for single pixel
  - Frame buffer
- Handle occlusion using depth buffer
  - Later details (more specifically, *fragments*) overwrite earlier ones if closer to camera
- Shade based on vertices and interpolate
  - See lighting and shading lecture

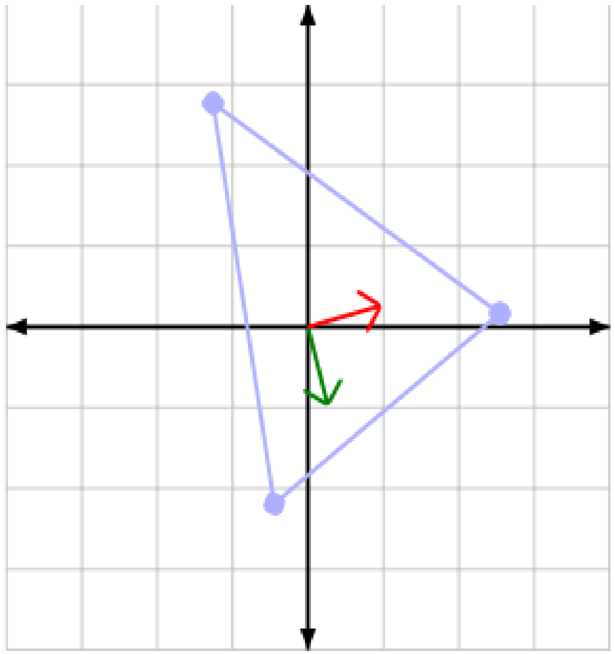
Geometry Transformations

Matrix multiplication

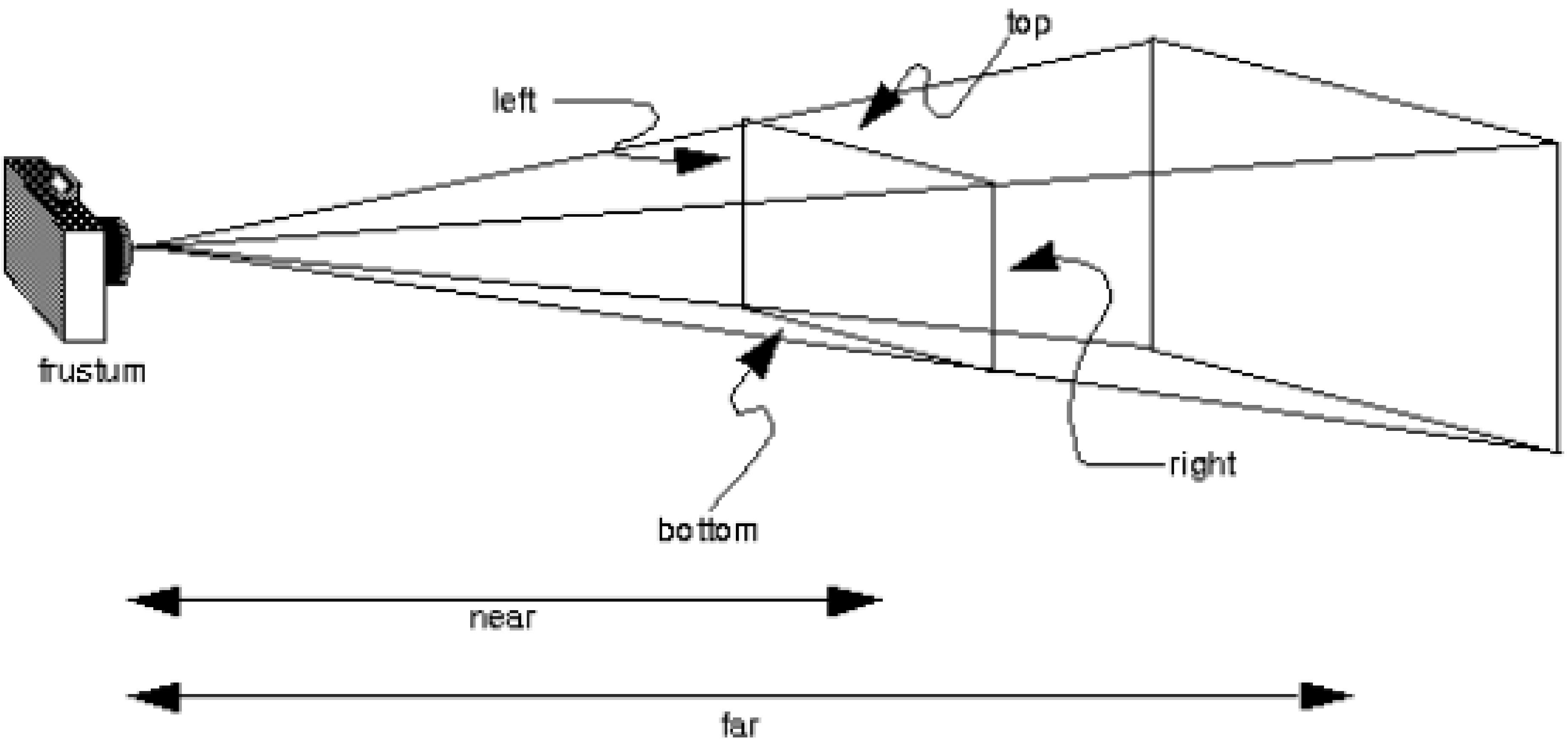
Translation, scaling, rotation, projection

Familiar?





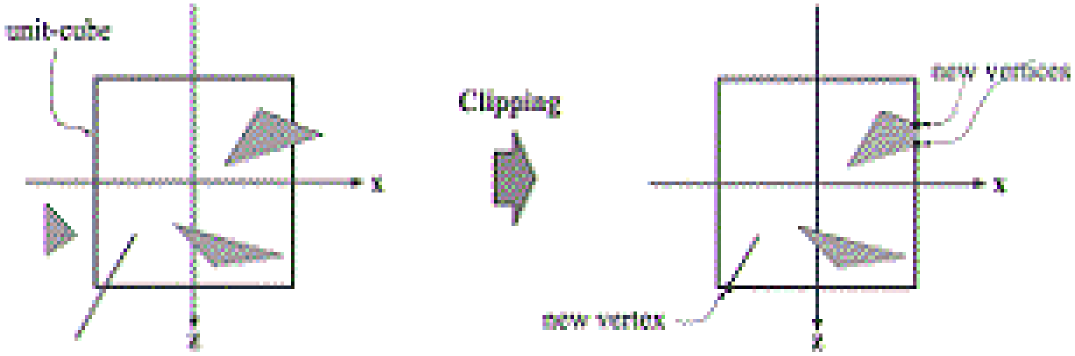
Perspective Projection

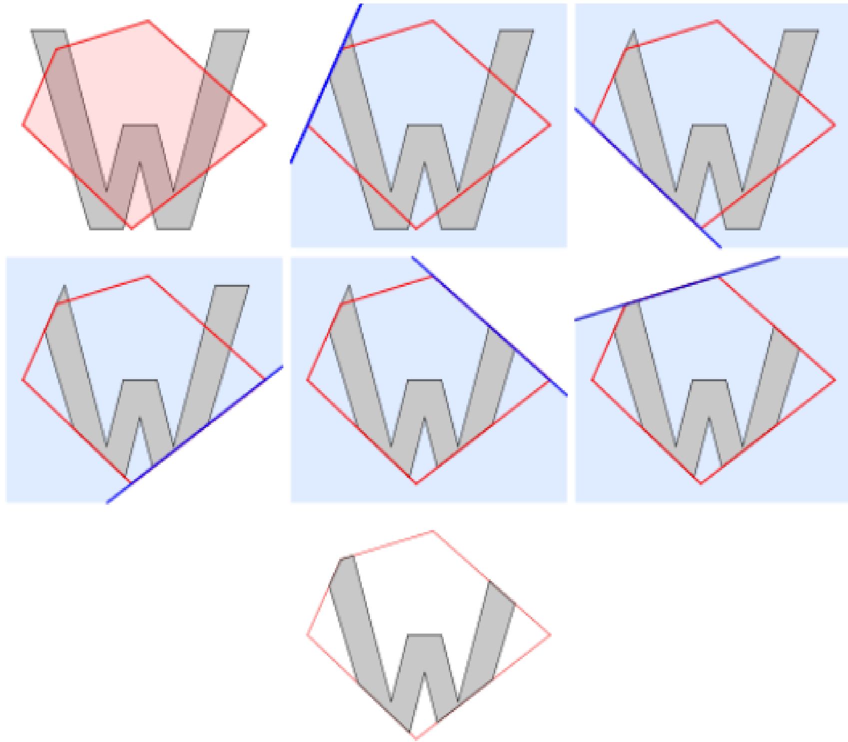


From OpenGL Programming Guide

Clipping

- Projected locations may be outside the viewing window
- Truncate triangles to fit them inside the viewing area
  - – e.g. Sutherland-Hodgeman algorithm

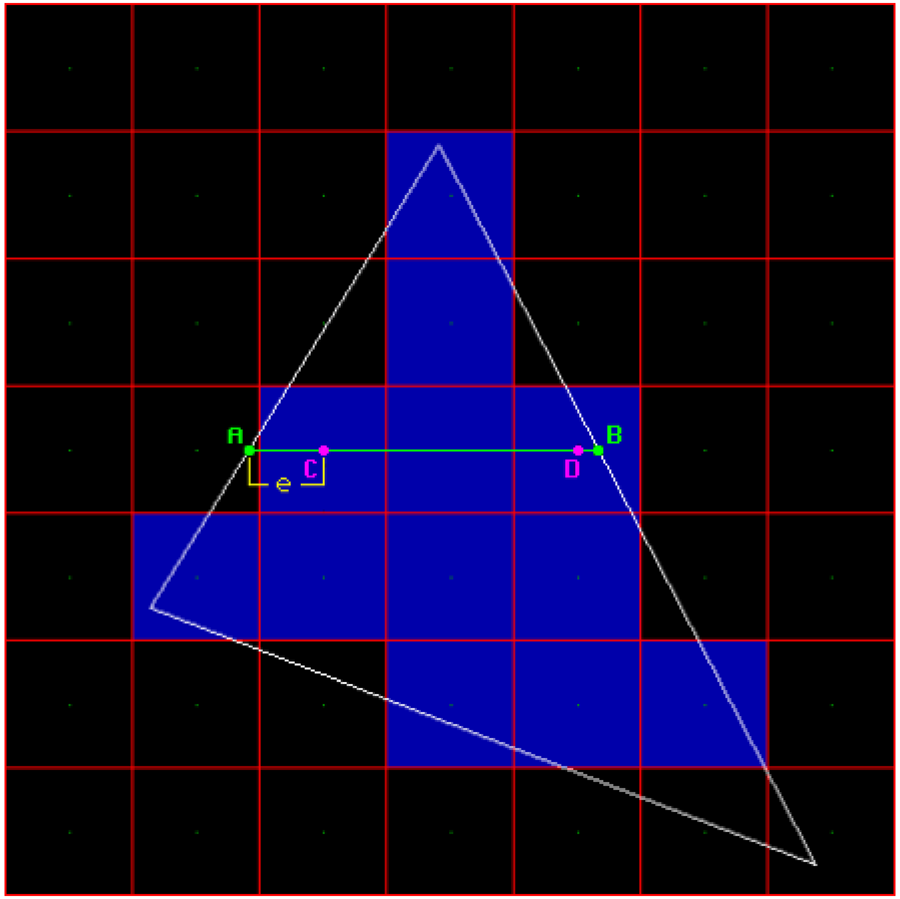




Real-time Rendering, Akenine-Moller, Haines and Hoffman

Scan Conversion

- Fill interior of triangles in image plane
- Use *scanline fill algorithm* to fill polygons
- Framebuffer



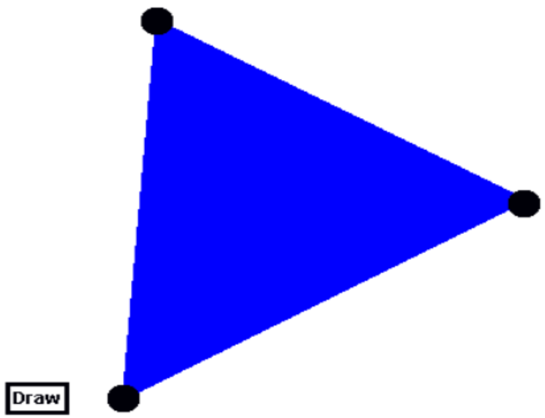
Polygon Filling

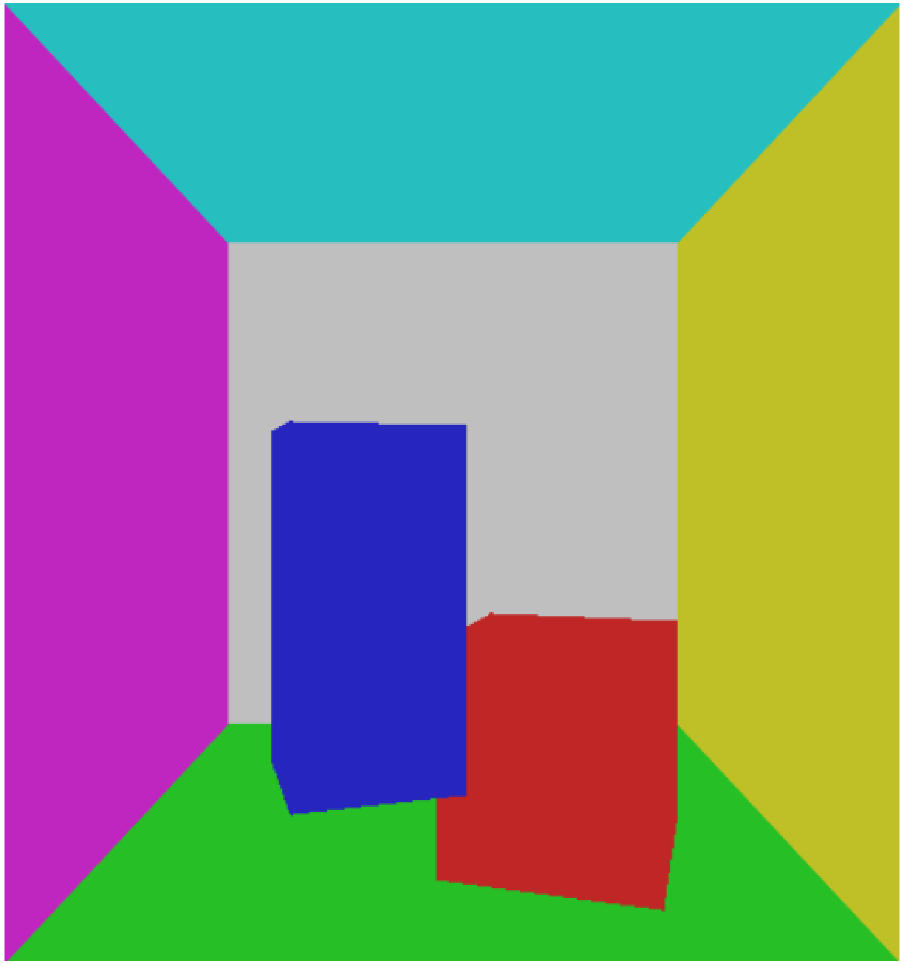
Fill surface

Triangles

Interpolation

Compute edges





Fixed Function Pipeline (FFP)

Beware: what follows is mostly deprecated

- Not programmable
  - Transform and rasterisation operations hardwired
- *Immediate mode* OpenGL
- Obsolete

Compare to modern *programmable pipeline*

- Vertex, pixel, etc shaders

Recommendation:

- Start with FFP to gain understanding of basics
- Learn programmable pipeline after

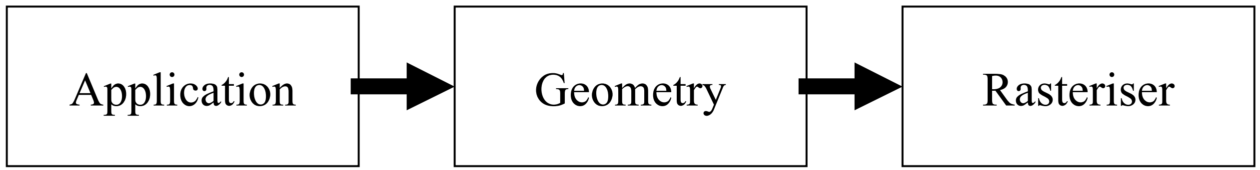
Graphics Pipeline Architecture

Can divide pipeline into three conceptual stages:

Application (input, animations, think SDL)

Geometry (transforms, projections, lighting)

Rasteriser (draw image as pixels)



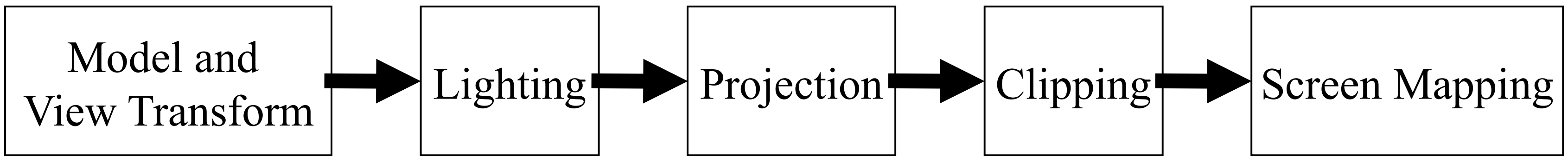
These define the core structure of the pipeline

Geometry Stage

Responsible for polygon and vertex operations

Consists of five sub-stages:

- Model and View Transform
- Lighting and Shading
- Projection
- Clipping
- Screen Mapping



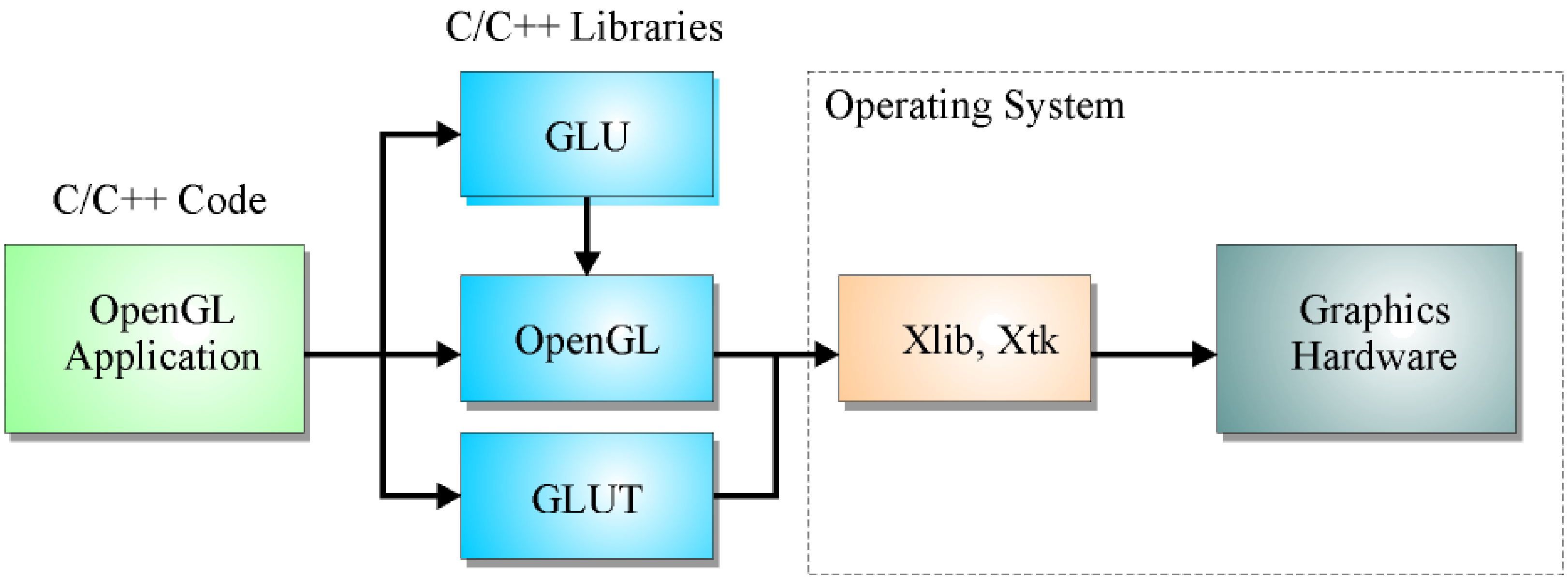
What is OpenGL ?

Software interface to graphics hardware

Commands for interactive three-dimensional graphics

Hardware independent interface

Drawing operations performed by underlying system and hardware



OpenGL Conventions

All function names begin with **gl**, **glu** or **glut**
- glVertex(…)
- gluSphere(…)
- glutMouseFunc(…)

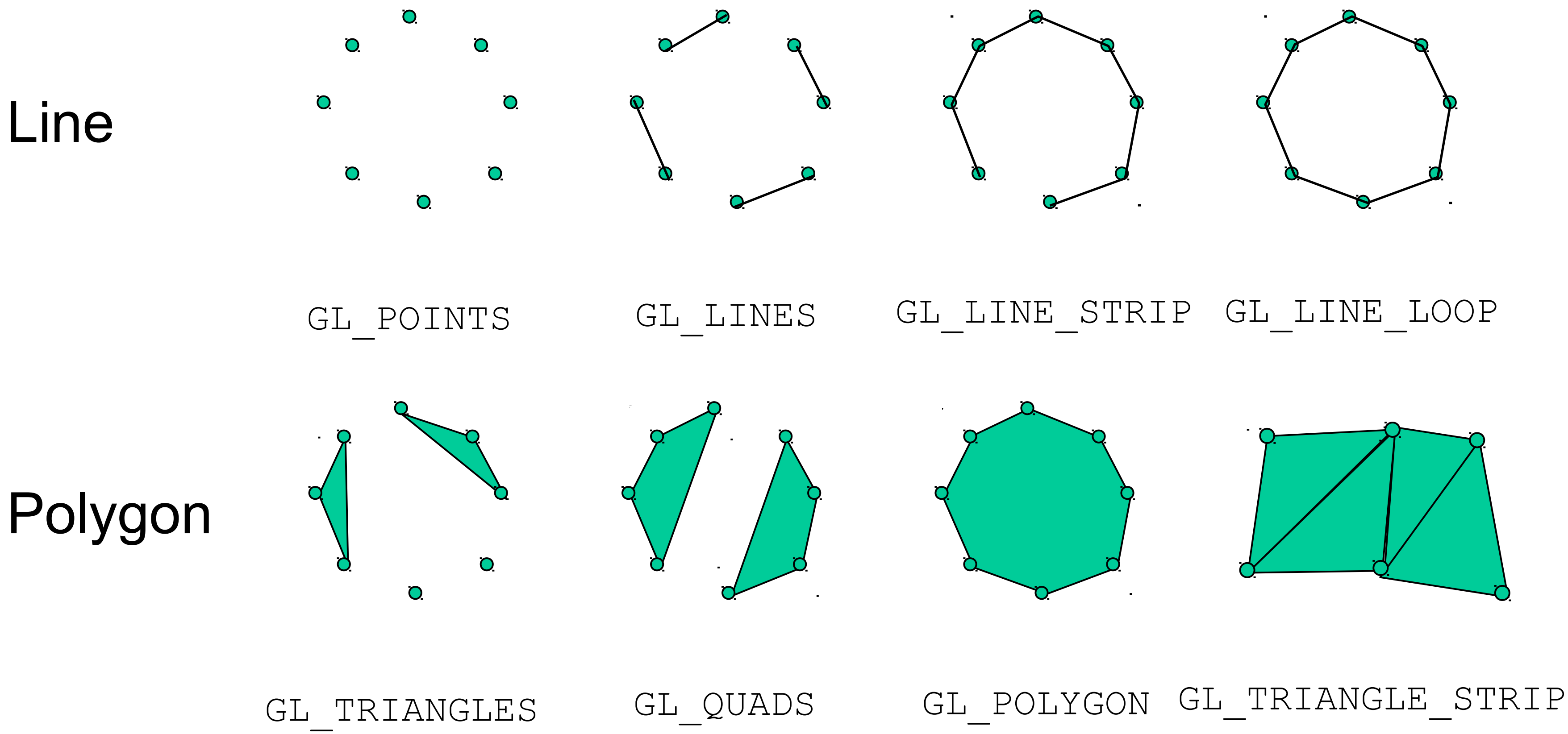
Constants begin with **GL_**, **GLU_** or **GLUT_**
- GL_FLOAT

Function names show parameter types
- glVertex**2i**(1, 3)
- glVertex**3f**(1.0, 3.0, 2.5)
- glVertex**4fv**(array_of_4_floats)

OpenGL Primitives

Basic building blocks

Line based primitives, and polygon based primitives:



Some OpenGL Code

To create a red polygon with 3 vertices:

```
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_POLYGON);
  glVertex3f(0.0, 0.0, 3.0);
  glVertex3f(1.0, 0.0, 3.0);
  glVertex3f(1.0, 1.0, 3.0);
glEnd();
```

glBegin defines the start of a new *geometric primitive*:

3D vertices are defined using **glVertex3***

The colour is defined using **glColor***

- Colours remain selected until changed
- OpenGL operates as a state machine

FreeGLUT

High level library for OpenGL like SDL

GLUT works using an *event loop:*

```
while (GLUT is running)
    e = get_next_event()
    if (e == mouse moved)
        run the mouse_moved function provided by the user
    if (e == redraw)
        run the draw function provided by the user
    etc, etc
```

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
glutCreateWindow("RGSquare Application");
glutReshapeWindow(400, 400);
```

RGB Colour, depth testing and double buffering

Window title

Request for window size (not necessarily accepted)

Glut Callback Functions

Some of the callback registration functions:

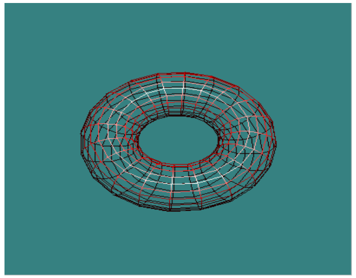
- ***void reshape(int w, int h) {…}***
- ***void keyhit(unsigned char c, int x, int y) {…}***
- ***void idle(void) {…}***
- ***void mouse(int button, int state, int x, int y) {…}***

`glutReshapeFunc(reshape);`	→	window resized
`glutKeyboardFunc(keyhit);`	→	key hit
`glutIdleFunc(idle);`	→	system idle
`glutDisplayFunc(draw);`	→	window exposed
`glutMotionFunc(motion);`	→	mouse moved
`glutMouseFunc(mouse);`	→	mouse button hit
`glutVisibilityFunc(visibility);`	→	window (de)iconified
`glutTimerFunc(timer);`	→	timer elapsed

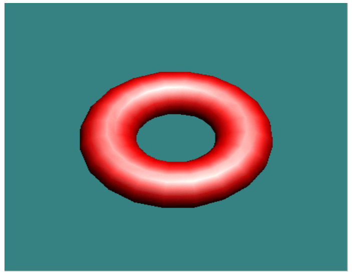
`glutMainLoop();` ← Begin event loop

Some Glut Objects

```
void glutSolidTorus(double inner_radius,
    double outer_radius, int nsides, int rings);
```



`glutWireTorus(0.3, 1.5, 20, 20);`



`glutSolidTorus(0.3, 1.5, 20, 20);`

`glutSolidTeapot(1.0);`

`glutSolidDodecahedron();`

Matrices in OGL

To initialise a matrix in OpenGL, use

```
glLoadIdentity();
```

Clears the currently selected OpenGL matrix (projection, transform, texture) to the *identity matrix*

Note: always be aware of what the matrix mode is activated!

To select a matrix as the current matrix, use

```
glMatrixMode(mode);
```

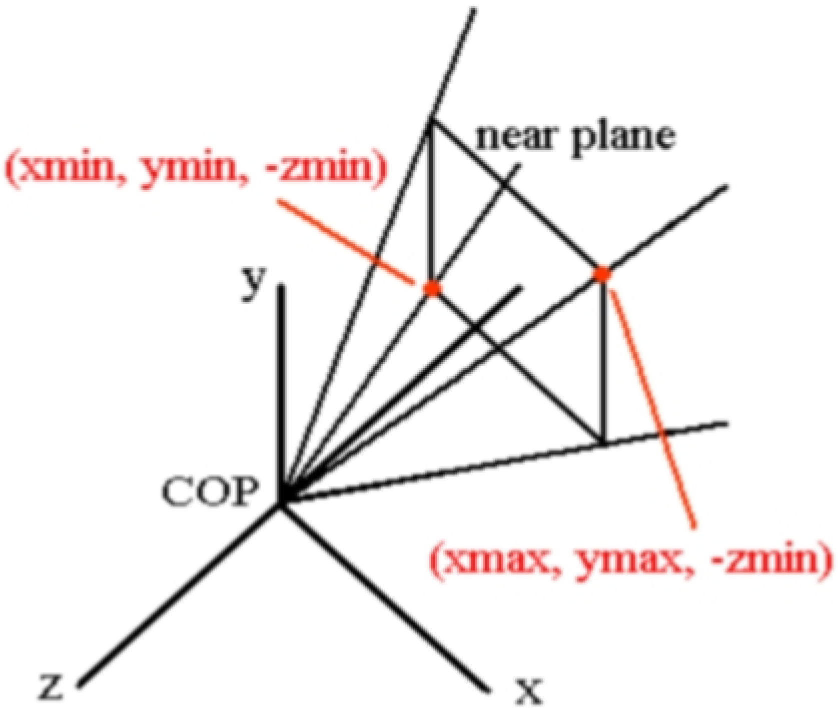
Mode can be one of:

GL_PROJECTION
GL_MODELVIEW
GL_TEXTURE

Perspective Projection

```
glFrustum(xmin, xmax, ymin, ymax, zmin, zmax);
```

- All points on the line defined by
  - (xmin, ymin, -zmin) and COP are mapped onto the lower left point on the viewport
  - (xmax, ymax, -zmin) and COP are mapped onto the upper right point on the viewport



Easier:

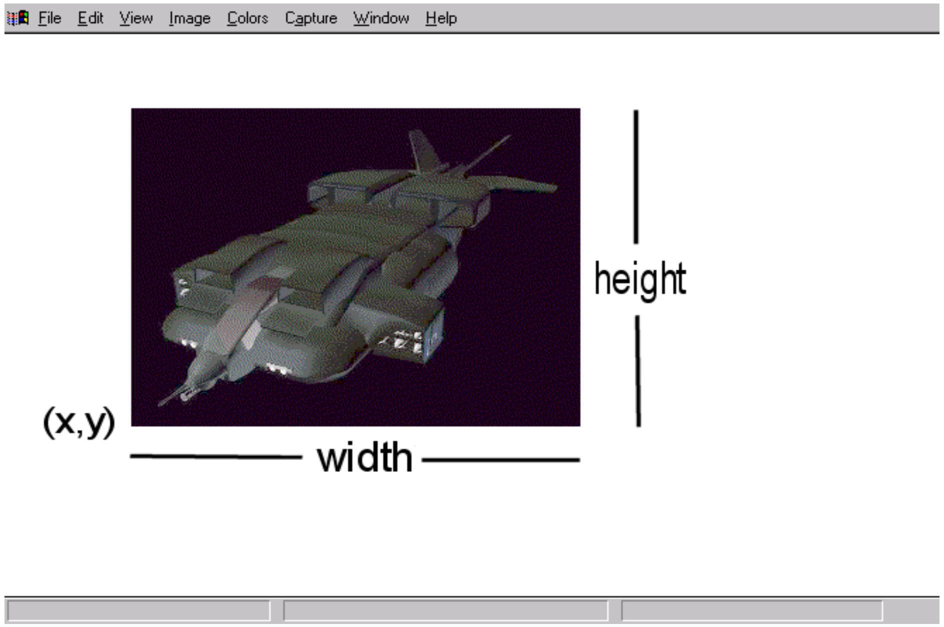
- `gluPerspective(fov, aspect, near, far);`

The Viewport

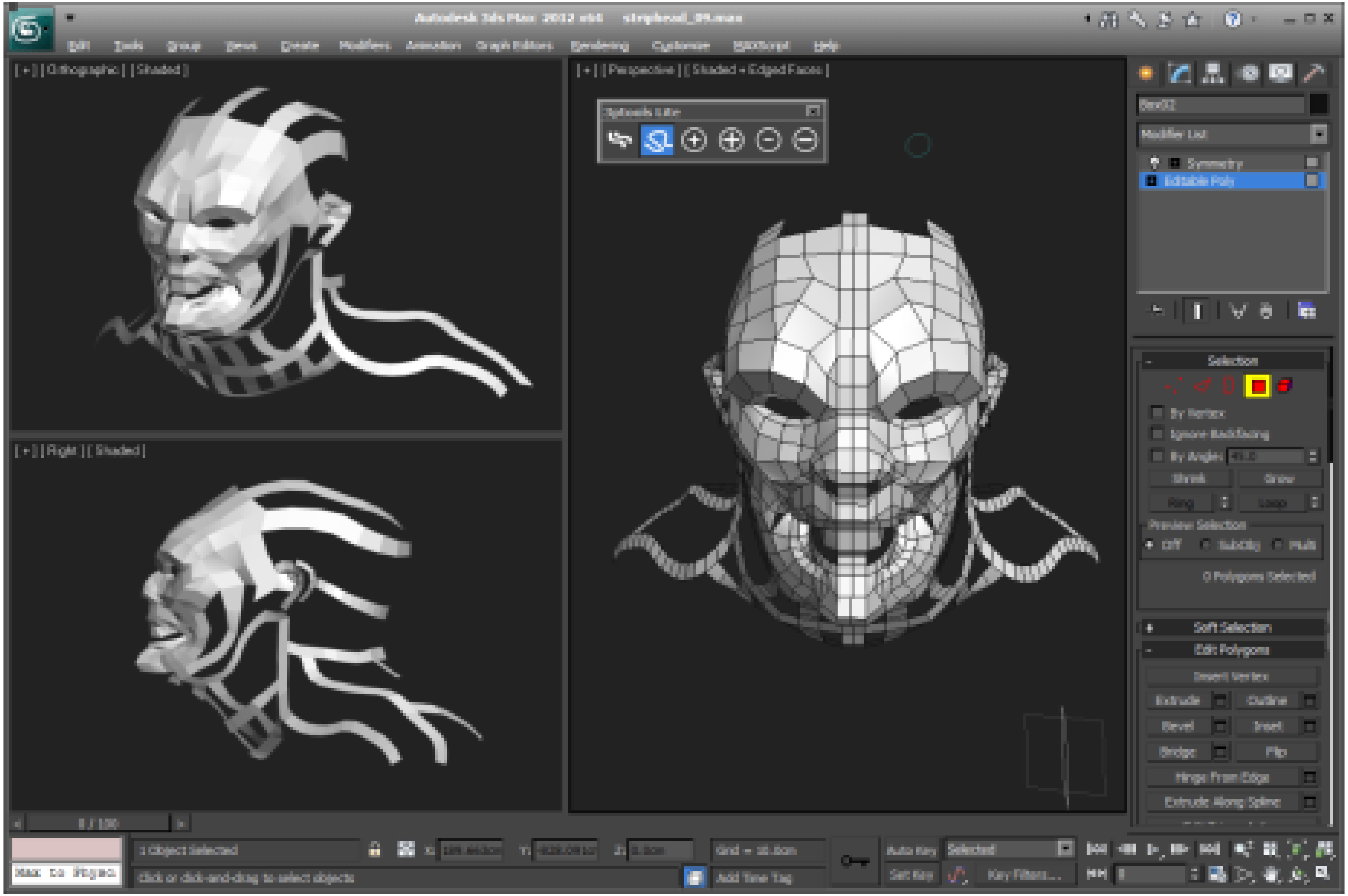
```
glViewport(x, y, width, height);
```

(x, y) is the location of the **bottom left** of the viewport within the window

- (0,0) is the bottom left hand side of the window
- *Width*, *height* specify the dimension in pixels of the viewport

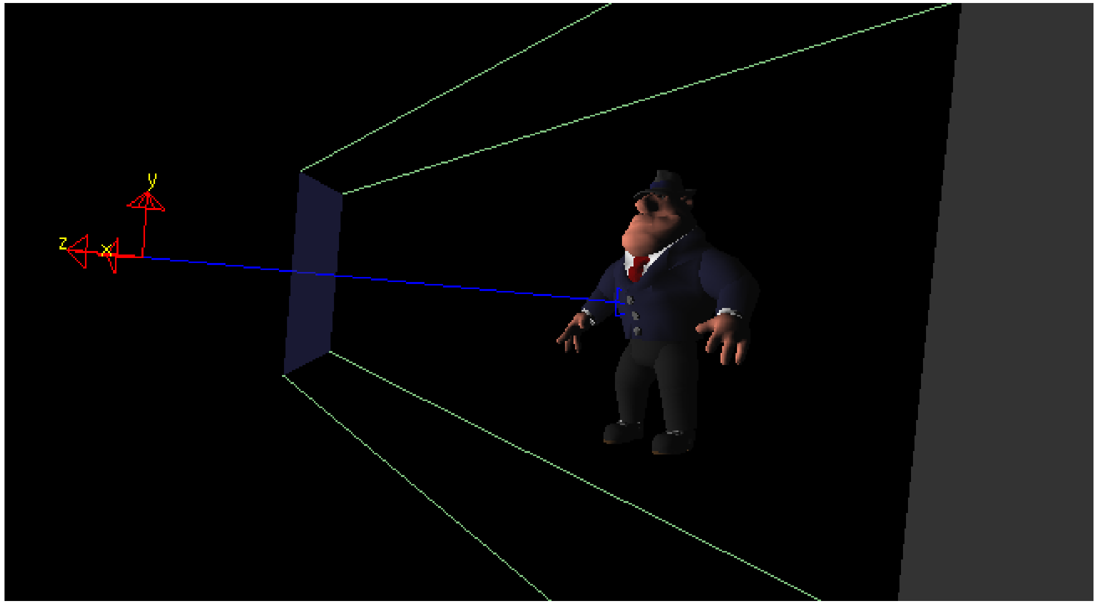


Useful for doing this...



3DS Max, 3ptools Lite, polycount

Positioning the Camera




```
gluLookAt(eye pos x, eye pos y, eye pos z, look at x, look at y, look at z, up x, up y, up z);
```

- Position camera at *eye pos (x,y,z)*
- Make camera viewing direction *look at (x,y,z)*
- Set camera up vector to *up (x,y,z)*

Projection and Camera

The following code will:

Create a viewport the size of the entire window

Create perspective projection with:

70 degree fovy, aspect ratio of 1, near plane at 1 and far plane at 50

Position the camera at (0.0,0.0,5.0) looking at (0.0,0.0,0.0)

```
glViewport(0,0,window_width, window_height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(70.0, 1.0, 1, 50);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
```

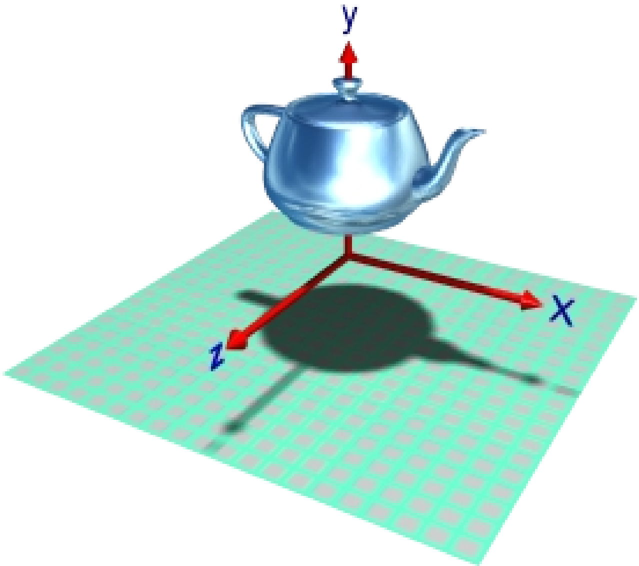
Transformations in OpenGL

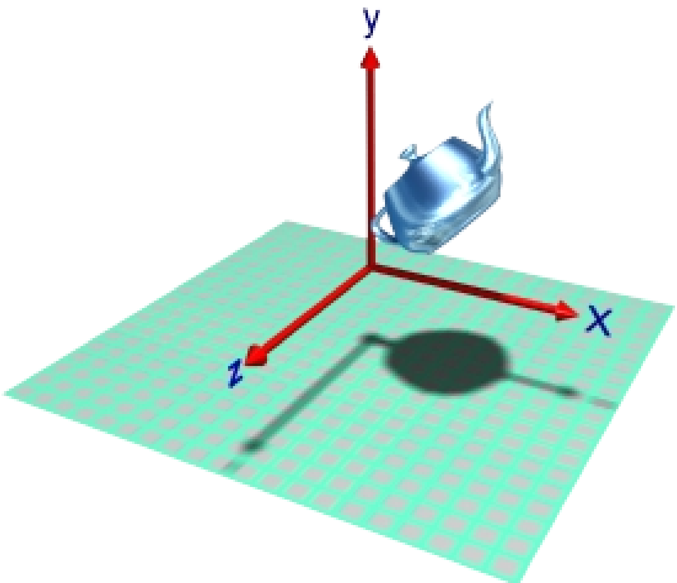
`glTranslate(dx, dy, dz);`

Translates by a displacement *(dx, dy, dz)*

Concatenates the specified translation to the current model-view matrix

Primitives drawn after this are modified by the specified translation





`glRotatef(angle, vx, vy, vz);`

Rotates around the axis *(vx, vy, vz)* by *angle* degrees

Concatenates the specified rotate to the current model-view matrix

Primitives drawn after this are modified by the specified rotation

ROYAL INSTITUTE OF TECHNOLOGY

Pushing and Popping

OpenGL allows us to push and pop matrices on a stack

- Remember the ***transform stack***
- ***`glPushMatrix();`***
- ***`glPopMatrix();`***

Pushing: save current local coordinate marker

Popping: retrieve/load previous local coordinate marker

There should always be the same number of matrix push operations as matrix pops

Why?

Example code

In order to draw two squares in different world-space positions:

```
glPushMatrix();
 glTranslate(square 1 WS position);
 glRotate(square 1 orientation);
 Draw_square();
glPopMatrix();

glPushMatrix();
 glTranslate(square 2 WS position);
 glRotate(square 2 orientation)
 Draw_square();
glPopMatrix();
```

Lighting and Shading

Enabling lighting: **`glEnable(GL_LIGHTING);`**

Select shading: **`glShadeModel(GL_SMOOTH);`**

Enable a light source: **`glEnable(GL_LIGHT1);`**

Specify parameters:

`glLightf{iv}(GL_LIGHT0, param, value);`

Material Properties

To assign material properties:

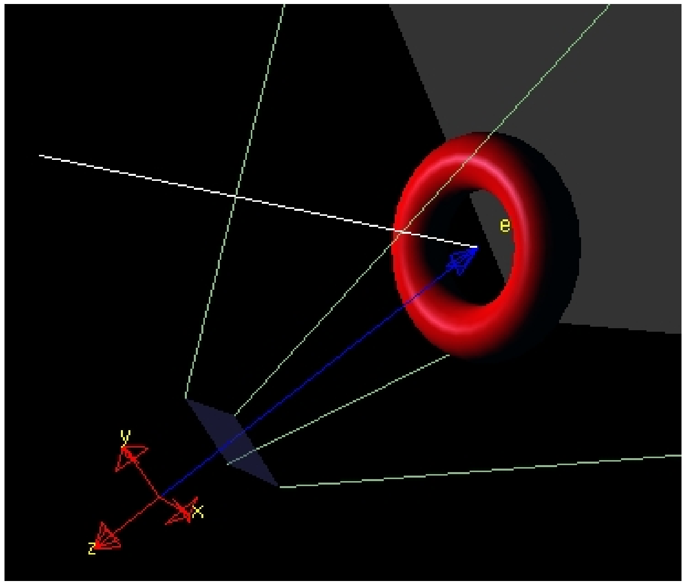
```
glMaterial{if}v(face, param, value);
```

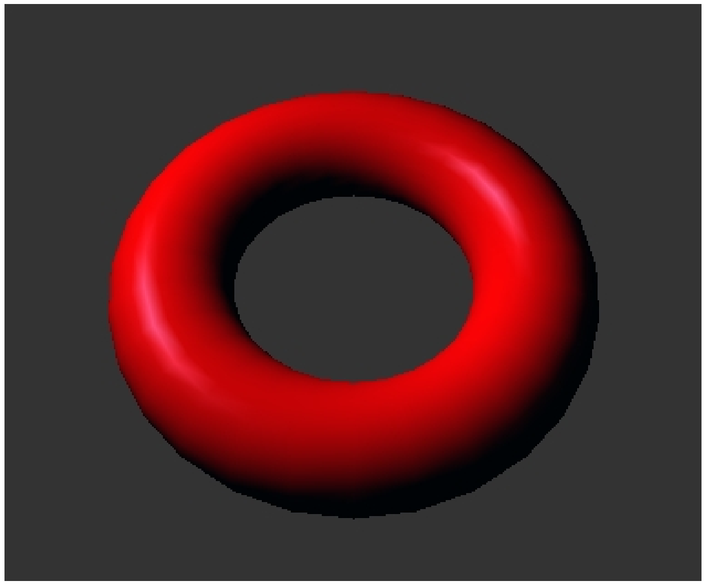
face:

```
GL_FRONT
GL_BACK
GL_FRONT_AND_BACK
```

param:

```
GL_AMBIENT
GL_DIFFUSE
GL_AMBIENT_AND_DIFFUSE
GL_SPECULAR
GL_EMISSION
GL_SHININESS
```





Conclusion

- Fixed function pipeline is deprecated
  - Older OpenGL versions
  - Useful to gain understanding of basics
- Compare to modern *programmable pipeline*
  - Vertex, pixel, etc shaders
  - Far more powerful

ROYAL INSTITUTE OF TECHNOLOGY

Links

http://nehe.gamedev.net

Legacy OGL tutorials (still great for learning)

http://www.khronos.org/webgl/

http://www.chromeexperiments.com/webgl/

WebGL, a graphics library for web browsers

Next lecture

- You should be working on Lab 3
- Next lab help session:

  13:00-15:00, Visualisation Studio, Thursday 5th May

- Animation and image based rendering
- Next week (9th May)
- 10:00 – 13:00