

# Graphics Pipeline in a Nutshell

CS334

Spring 2008

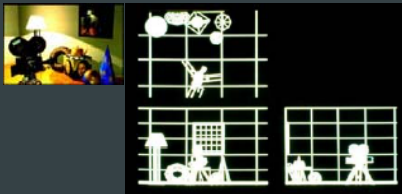
## Computer Graphics Pipeline

- How do we create a rendering such as this?



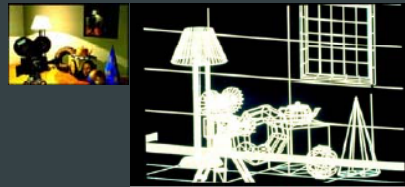
## Computer Graphics Pipeline

- Design the scene (technical drawing in "wireframe")



## Computer Graphics Pipeline

- Apply perspective transformations to the scene geometry for a virtual camera



## Computer Graphics Pipeline

- Hidden lines removed and colors added



## Computer Graphics Pipeline

- Geometric primitives filled with constant color



### Computer Graphics Pipeline

- View-independent lighting model added

### Computer Graphics Pipeline

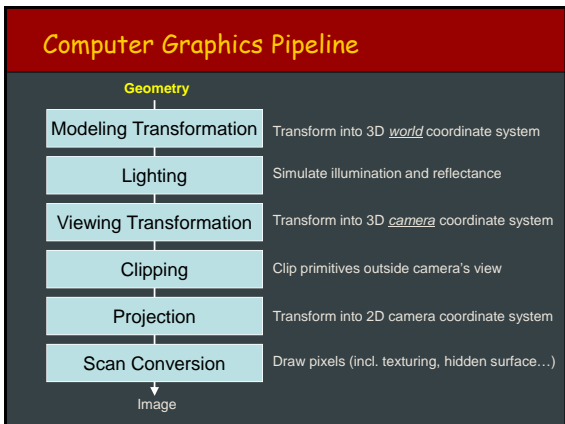
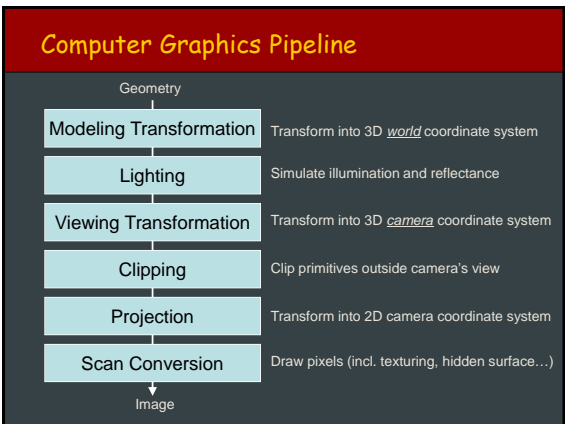
- View-dependent lighting model added

### Computer Graphics Pipeline

- Texture mapping: pictures are wrapped around objects

### Computer Graphics Pipeline

- Reflections, shadows, and bumpy surfaces



## Geometry

- How are the objects described in a computer?
  - Points (or vertices)
    - e.g., x, y, z
  - Lines
    - $l(t) = a+tb$
    - (a,b)
    - (u,v,s,t)
  - Triangles
    - 3 points
    - 2 vectors (cross product)
    - meshes: nice collection of triangles
    - strips: efficient collection of connected triangles
  - Polygons
    - n-sided polygons
    - typically convex (why?)
    - polygon soup
  - Manifold:
    - 2D manifold in 3D space: what are they good for?
      - Is a tabletop object part of the table or a separate object?
  - Curved surfaces
    - B-spline patches
  - Functions
    - $F(x,y,z)=0$
    - Superquadrics, etc

## Geometry

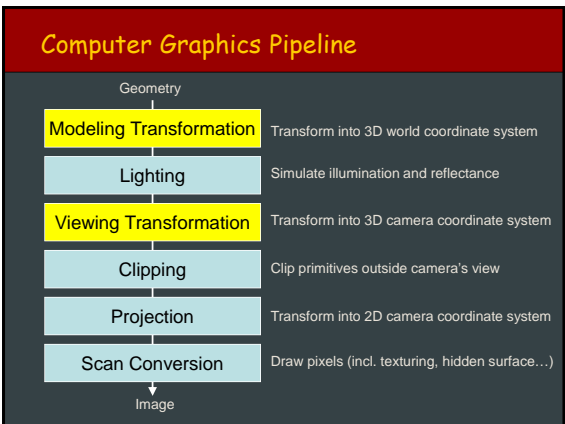
- What else?
- Geometry image:
  - <http://research.microsoft.com/users/hhoppe/qim.pdf>

## Geometry

- What information is needed per vertex?
  - Position
  - Normal
  - Color
  - Texture coordinates...

## Geometry

- What information is needed per geometric primitive?
  - Color
  - Normal
  - Material properties (e.g. textures...)



## Transformations

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

Identity

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Mirror over X axis

## Transformations

Rotate around Z axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

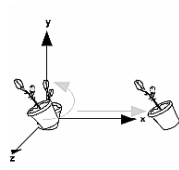
Rotate around Y axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotate around X axis:

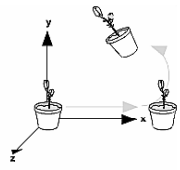
$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

## Transformations



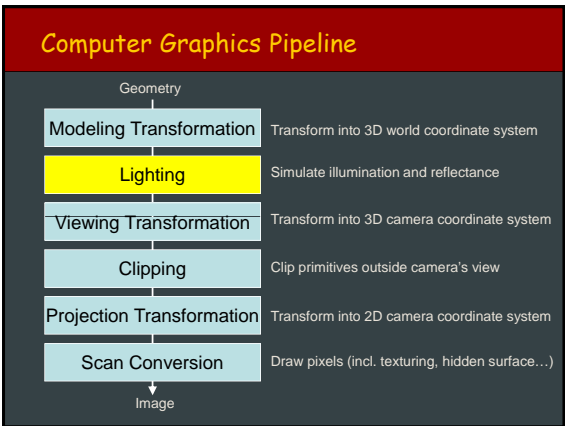
Rotate then Translate

```
glRotatef(d,rx,ry,rz);
glTranslate3f(tx,ty,tz);
```

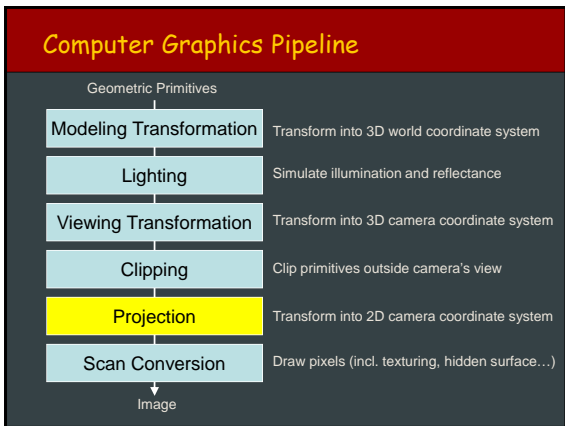
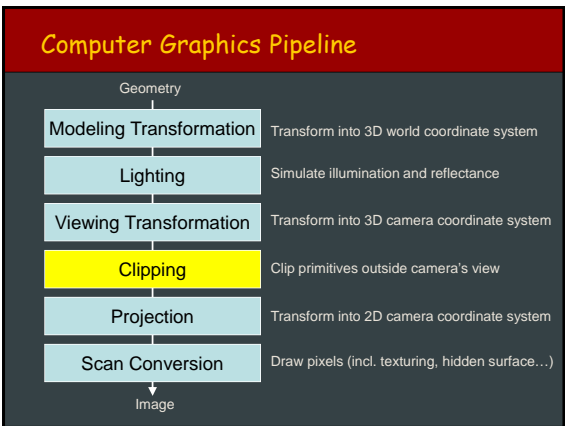


Translate then Rotate

```
glTranslate3f(tx,ty,tz);
glRotatef(d,rx,ry,rz);
```



- ## Lighting and Shading
- Light sources
    - Point light
      - Models an omnidirectional light source (e.g., a bulb)
    - Directional light
      - Models an omnidirectional light source at infinity
    - Spot light
      - Models a point light with direction
  - Light model
    - Ambient light
    - Diffuse reflection
    - Specular reflection



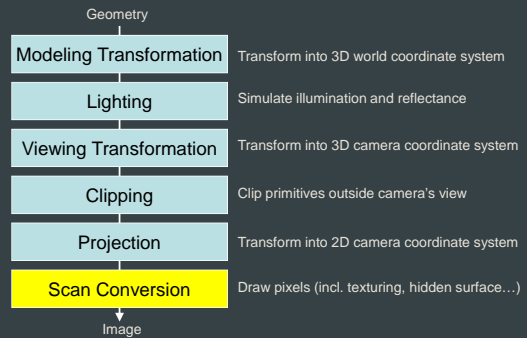
## Projections

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ \frac{0}{r-l} & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspective projection

■ Etc...

## Computer Graphics Pipeline



## OpenGL

- Software interface to graphics hardware
- ~150 distinct commands
- Hardware-independent and widely supported
  - To achieve this, no windowing tasks are included
- GLU (Graphics Library Utilities)
  - Provides some higher-level modeling features such as curved surfaces, objects, etc.
- Open Inventor
  - A higher-level object-oriented software package

## OpenGL Online

- Programming Guide ("Red book")
  - [http://www.opengl.org/documentation/red\\_book\\_1.0](http://www.opengl.org/documentation/red_book_1.0)
- Reference Manual ("Blue book")
  - [http://www.opengl.org/documentation/blue\\_book\\_1.0](http://www.opengl.org/documentation/blue_book_1.0)

## Simple OpenGL Program

```
{
  <Initialize OpenGL state>

  <Load and define textures>

  <Specify lights and shading parameters>

  <Load projection matrix>

  For each frame
    <Load model view matrix>
    <Draw primitives>
  End frame
}
```

## Matrix Transformations

- Each of modelview and projection matrix is a 4x4 matrix
- OpenGL functions
  - glMatrixMode(...)
  - glLoadIdentity(...)
  - glLoadMatrixf(...)
  - glMultMatrix(...)
  - glTranslate(...)
  - glScale(...)
  - glRotate(...)
  
  - glPushMatrix()
  - glPopMatrix()

## Matrix Transformations

```

{
-
-
-
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N); /* apply transformation */
glMultMatrixf(M); /* apply transformation M */
glMultMatrixf(L); /* apply transformation L */
glBegin(GL_POINTS);
glVertex3f(v); /* draw transformed vertex v */
glEnd();
-
-
}

= draw transformed point "N(M(Lv))"

```

## Simple Program

```

#include <GL/gl.h>
main()
{
    InitializeWindowPlease();
    glMatrixMode(GL_PROJECTION);
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.0);
        glVertex3f(0.75, 0.25, 0.0);
        glVertex3f(0.75, 0.75, 0.0);
        glVertex3f(0.25, 0.75, 0.0);
    glEnd();
    glFlush();
    UpdateTheWindowAndCheckForEvents();
}

```

## GLUT

- = Graphics Library Utility Toolkit
  - Adds functionality such as windowing operations to OpenGL
- Event-based callback interface
  - Display callback
  - Resize callback
  - Idle callback
  - Keyboard callback
  - Mouse movement callback
  - Mouse button callback

## Simple OpenGL + GLUT Program

```

#include <...>
DisplayCallback()
{
    <Clear window>
    <Load Projection matrix>
    <Load Modelview matrix>
    <Draw primitives>
    <Swap buffers>
}

IdleCallback()
{
    <Do some computations>
    <Maybe force a window refresh>
}

KeyCallback()
{
    <Handle key presses>
}

MouseMovementCallback()
{
    <Handle mouse movement>
}

MouseButtonsCallback()
{
    <Handle mouse buttons>
}

Main()
{
    <Initialize GLUT and callbacks>
    <Create a window>
    <Initialize OpenGL state>

    <Enter main event loop>
}

```

## Simple OpenGL + GLUT Program

```

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glLoadIdentity();
    gluLookAt(0, 0, 5, 0, 0, 0, 1, 0);
    glScalef(1.0, 2.0, 1.0);
    glutWireCube(1.0);
    glFlush();
}

void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}

```

## Example Program with Lighting

```

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

void init(void)
{
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);

    void display(void)
    {
        glClear(GL_COLOR_BUFFER_BIT |
                GL_DEPTH_BUFFER_BIT);
        glutWireSphere(1.0, 20, 16);
        glFlush();
    }

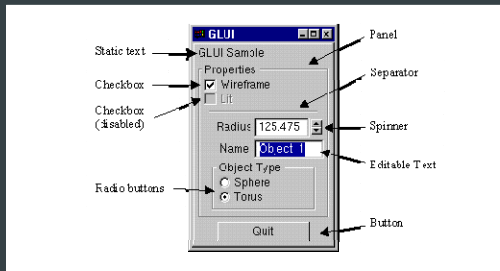
    void reshape(int w, int h)
    {
        glViewport(0, 0, (GLsizei) w, (GLsizei) h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if (w <= h)
            glOrtho(-1.5, 1.5, -1.5, 1.5, -10.0, 10.0);
        else
            glOrtho(-1.5*(GLfloat)w/(GLfloat)h,
                    1.5*(GLfloat)w/(GLfloat)h, -10.0, 10.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        int main(int argc, char** argv)
        {
            glutInit(&argc, argv);
            glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB |
                                GLUT_DEPTH);
            glutInitWindowSize(500, 500);
            glutInitWindowPosition(100, 100);
            glutCreateWindow(argv[0]);
            init();
            glutDisplayFunc(display);
            glutReshapeFunc(reshape);
            glutMainLoop();
            return 0;
        }
    }
}

```

## GLUI

- = Graphics Library User Interface



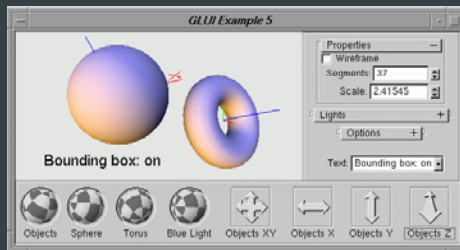
## GLUI

- = Graphics Library User Interface



## GLUI

- = Graphics Library User Interface



## Alternative graphics pipeline?

- Traditional pipeline...ok
- Parallel pipeline
  - Cluster of PCs?
  - Cluster of PS3?
  - What must be coordinated? What changes? What are the bottlenecks?
- Sort-first vs. Sort-last pipeline
  - PixelFlow
  - Several hybrid designs