Computer Graphics in a Nutshell

CS 635
Spring 2009

History and applications
- Computer Graphics Pipeline
- Linear Algebra Review
  - Vectors, points
  - Matrices, transformations
- Representations
  - Points, lines, polygons, objects, meshes
  - Textures and images
- Lighting and Shading
  - Flat, Gouraud, Phong
- Some advanced topics
  - Global illumination
  - Ray tracing
  - Antialiasing

(Some slides courtesy of Thomas Funkhouser and Marcus Magnor)

Computer Graphics II

- OpenGL
  - Motivation
  - Graphics context/state
  - Basic program outline
- Rendering geometric primitives
  - Points, lines, polygons
- Lighting and Shading
  - Flat, Gouraud, Phong
- Texturing Polygons
- GLUT
  - Multiple windows and contexts
  - User input
  - Event loops
- GLUI
  - A simple Graphical User Interface designer

History and Applications

- 1955: MIT Whirlwind (CRT)
- 1955: Sage, Radar with CRT and light pen
- 1958: Willy Higinbotham “Tennis”
- 1960: MIT “Spacewar” on DEC-PDP-1
- 1963: Ivan Sutherland’s “Sketchpad” (CAD)
- 1968: Tektronix storage tube
- 1968: Evans & Sutherland’s flight simulators
- 1968: Douglas Engelbart: computer mouse
- 1969: ACM SIGGRAPH
- 1970: Xerox, Gull
- 1971: Gouraud shading
- 1974: Z-buffer
- 1975: Phong Model
- 1979: Eurographics
- 1981: Apollo Workstation, PC
- 1982: Whitt, Ray tracing
- 1982: SGI
- 1984: X Window System
- 1984: 1st SGI Workstation
- ->1995: SGI dominance
- ->2003: PC dominance
- Today: programmable graphics hardware (again)

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

Computer Graphics Pipeline

<table>
<thead>
<tr>
<th>Geometric Primitives</th>
<th>Transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling Transformation</td>
<td>Transform into 3D world coordinate system</td>
</tr>
<tr>
<td>Lighting</td>
<td>Simulate illumination and reflectance</td>
</tr>
<tr>
<td>Viewing Transformation</td>
<td>Transform into 3D camera coordinate system</td>
</tr>
<tr>
<td>Clipping</td>
<td>Clip primitives outside camera's view</td>
</tr>
<tr>
<td>Projection Transformation</td>
<td>Transform into 2D camera coordinate system</td>
</tr>
<tr>
<td>Scan Conversion</td>
<td>Draw pixels (incl. texturing, hidden surface...)</td>
</tr>
</tbody>
</table>

Linear Algebra Review

- Why do we need it?
  - Modeling transformation
    - Move "objects" into place relative to a world origin
  - Viewing transformation
    - Move "objects" into place relative to camera
  - Perspective transformation
    - Project "objects" onto image plane

Transformations

- Most popular transformations in graphics
  - Translation
  - Rotation
  - Scale
  - Projection
- In order to use a single matrix for all, we use homogeneous coordinates...
Transformations

How are the objects described in a computer?
- Points (or vertices)
- Lines
- Triangles
- Polygons
- Curved surfaces, etc.
- Functions

Representations

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

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

Rotations around axes:
- Identity
- Scale
- Translation
- Mirror over X axis

Perspective projection

Exercise
Texture Mapping

- Map a "texture" onto the surface of an object
  - Wood, marble, or any "pattern"

Texture Mapping

- A texture is a two-dimensional array of "texels", indexed by a (u,v) texture coordinate
- At each screen pixel, a texel can be used to substitute a geometric primitives surface color

Texture Mapping

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

Lighting and Shading

- Diffuse reflection
  - Lambertian model

\[ I_D = K_D (N \cdot L) I_L \]
Lighting and Shading
- Specular reflection
  - Phong model

\[ I_g = K_g (V \cdot R)^m I_i \]

...shadows?

Advanced Topics: Global Illumination

Advanced Topics: Ray tracing

Advanced Topics: Antialiasing
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/blue_book_1.0

OpenGL

- Rendering parameters
  - Lighting, shading, lots of little details...
- Texture information
  - Texture data, mapping strategies
- Matrix transformations
  - Projection
  - Model view
  - (Texture)
  - (Color)

Matrix Transformations

- Each of modelview and projection matrix is a 4x4 matrix
- OpenGL functions
  - glMatrixMode(…)
  - glLoadIdentity(…)
  - glLoadMatrixf(…)
  - glMultMatrix( )
  - glTranslatef(…)
  - glScalef(…)
  - glRotatef(…)
  - glPushMatrix()
  - glPopMatrix()
Matrix Transformations

```c
{...
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glMultMatrixf(N); /* apply transformation */
  glMultMatrixf(M); /* apply transformation M */
  glMultMatrixf(L); /* apply transformation L */
  glUseProgram();
  glVertex3f(v); /* draw transformed vertex v */
  glEnd();
}
= draw transformed point "N(M(Lv))"
```

Modelview Transformations

```c
void pilotView(GLdouble planex, GLdouble planey, GLdouble
planez, GLdouble roll, GLdouble pitch, GLdouble heading)
{
  glRotated(roll, 0.0, 0.0, 1.0);
  glRotated(pitch, 0.0, 1.0, 0.0);
  glRotated(heading, 1.0, 0.0, 0.0);
  glTranslated(-planex, -planey, -planez);
}
void polarView(GLdouble distance, GLdouble twist, GLdouble
elevation, GLdouble azimuth)
{
  glTranslated(0.0, 0.0, -distance);
  glRotated(-twist, 0.0, 0.0, 1.0);
  glRotated(-elevation, 1.0, 0.0, 0.0);
  glRotated(azimuth, 0.0, 0.0, 1.0);
}
```

Projection Transformations

```c
void glFrustum(GLdouble left, GLdouble right, GLdouble
bottom, GLdouble top, GLdouble near, GLdouble far);
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble
near, GLdouble far);
```
Projection Transformations

```c
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);
```

Matrix Transformations

```c
draw_wheel_and_bolts()
{
   long i;
draw_wheel();
   for(i=0;i<5;i++)
   {
      glPushMatrix();
      glPopMatrix();
   }
}
```

Simple OpenGL Program

```c
#include <GL/gl.h>
main()
{    
   InitializeOpenGLState();
   LoadAndDefineTextures();
   SpecifyLightsAndShadingParameters();
   LoadProjectionMatrix();

   For each frame
   {      
      LoadModelViewMatrix();
      DrawPrimitives();
   }

   End frame
}
```

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

```c
#include <GL/gl.h>
DisplayCallback()
{    
   ClearWindow();
   LoadProjectionMatrix();
   LoadModelViewMatrix();
   DrawPrimitives();
   SwapBuffers();
}
IdleCallback()
{    
   DoSomeComputation();
   ForceWindowRefresh();
}
KeyCallback()
{    
   HandleKeyPress();
}
MouseMovementCallback()
{    
   HandleMouseMovement();
}
MouseButtonsCallback()
{    
   HandleMouseButtons();
}
Main()
{    
   InitializeGLUTAndCallBacks();
   CreateAWindow();
   InitializeOpenGLState();
   <Create main event loop>
}
```
Simple OpenGL + GLUT Program

```c
#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 reshape (int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
    glFrustum (-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;
}
```

display()

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

Example Program with Lighting

```c
#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);
}
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*(GLfloat)h/(GLfloat)w,
            1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
    else
        glOrtho (-1.5*(GLfloat)w/(GLfloat)h,
            1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
}
void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutSolidSphere (1.0, 20, 16);
    glFlush ();
}
```

main()

```c
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;
}
```