Rendering Pipeline/OpenGL

Your tasks for the weekend

- Piazza Discussion Group:
  - Register
  - Post review questions by Mon noon
    - Use private option, rev1 tag
- Start Assignment 1
  - Test programming environment on lab computers/Set laptop environment (optional)

Assignment 1

- Experience OpenGL & GLUT
  - Have FUN
  - Deadline: Sep 20

Your tasks for the weekend

- Sign and Submit Plagiarism Form
- Optional reading (Shirley: Introduction to CG)
  - Math refresher: Chapters 2, 4
    - Lots of math coming in the next few weeks
  - Background on graphics: Chapter 1

Rendering

Goal:
- Transform (3D) computer models into images
- Photo-realistic (or not)
Interactive rendering:
- Fast, but (until recently) low quality
- Roughly follows a fixed pattern of operations
  - Rendering Pipeline
Offline rendering:
- Ray-tracing
- Global illumination

Rendering Tasks (no particular order)

- Project 3D geometry onto image plane
  - Geometric transformations
- Determine which primitives/parts of primitives are visible
  - Hidden surface removal
- Determine which pixels geometric primitive covers
  - Scan conversion
- Compute color of every visible surface point
  - Lighting, shading, texture mapping

Copyright  A. Sheffer, 2013, UBC
The Rendering Pipeline

- Abstract model of
  - sequence of operations to transform geometric model into digital image
  - graphics hardware workflow
- Underlying API (application programming interface) model for programming graphics hardware
  - OpenGL
  - Direct 3D
- Actual implementations vary

Clicker Question

- Which of the tasks below is not part of the rendering pipeline?
  - A. Scan Conversion
  - B. Viewing Transformation
  - C. Modeling
  - D. Lighting

(Tentative) Lecture Syllabus

- Introduction + Rendering Pipeline (week 1/2)
- Transformations (week 2/3)
- Scan Conversion (week 4/5)
- Clipping (week 5)
- Hidden Surface Removal (week 6/7)
- Review & Midterm (week 7)
  - Midterm: Oct 18
- Lighting Models (week 8)
- Texture mapping (week 9/10)
- Review & Midterm (week 10)
  - Midterm: Nov 8
- Ray Tracing (week 11)
- Shadows (week 11/12)
- Modeling (content creation) (week 12/13)
- Review (last lecture)

OpenGL

- API for graphics hardware
  - Started in 1989 by Kurt Akeley
  - Designed to exploit graphics hardware
  - Implemented on many different platforms
- Pipeline processing
  - Event driven
  - Communication via state setting
Event-driven Programming

- Main loop not under your control
  - vs. procedural
- Control flow through event callbacks
  - redraw the window now
  - key was pressed
  - mouse moved
- Callback functions called from main loop
  - when events occur
  - mouse/keyboard, redrawing...

OpenGL/GLUT Example

```c
void display(void) {  // Called when need to redraw screen.
    // Clear the buffer we will draw into.
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);

    // Initialize the modelview matrix.
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Draw STUFF
    // Make the buffer we just drew into visible.
    glutSwapBuffers();
}
```

GLUT Example

```c
int main(int argc, char **argv) {  
    ....
    // Schedule the first animation callback ASAP.
    glutTimerFunc(0, animate, 0);
    // Pass control to GLUT.
    glutMainLoop();
    return 0;
}
```

GLUT Input Events

```c
void reshape(int w, int h) {
    ....
    // you supply these kind of functions
}
void keyboard(unsigned char key, int x, int y) {
    ....
    // register them with glut
    glutKeyboardFunc(keyboard);
}
void mouse(int but, int state, int x, int y) {
    ....
}
```
**GLUT and GLU primitives**

- `gluSphere()`
- `glutSolidSphere(GLdouble radius, GLint slices, GLint stacks)`
- `glutWireSphere()`
- `glutSolidCube(GLdouble size)`
- `glutWireCube()`
- `glutSolidTorus()`
- `glutWireTorus()`
- `glutSolidTeapot()`
- `glutWireTeapot()`

**Example (from a1):**

```cpp
void Turtle::draw() {
    glPushMatrix();
    glTranslatef(x_, y_, 0);
    // Turtle shell.
    glColor4fv(shell_);
    glBegin(GL_POLYGON);
        for (double i = 0; i < M_PI; i += M_PI / 12)
            glVertex3f(cos(i) * radius_, sin(i) * radius_, 0.0);
    glEnd();
    …..
    glPopMatrix();
}
```

**GLUT and GLU primitives**

- Basic Transformations:
  - `glTranslatef(…);`
  - `glRotatef(…);`
  - `glScalef(…);`

**Lighting**

```cpp
void setup_lighting(void) {
    // Turn on lighting, and two local lights.
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHT1);
    glEnable(GL_COLOR_MATERIAL);
    // Set the intensity of the global ambient light.
    float ambient[] = {0.3, 0.3, 0.3, 1.0};
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambient);
    // Set up the diffuse intensities of the local light source.
    float diffuse[] = {
        0.8, 0.8, 0.8, 1.0,
        0.2, 0.2, 0.2, 1.0,
    };
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse[0]);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuse[1]);
    // Move the light near the top corner of the window.
    float light_positions[] = {
        0,  1, 2, 0, // From above-left
        0, -5, 0, 0, // From below
    };
    glLightfv(GL_LIGHT0, GL_POSITION, light_positions[0]);
    glLightfv(GL_LIGHT1, GL_POSITION, light_positions[1]);
}
```

**Clicker Question**

- What does the function ‘glutMainLoop’ do?
  - a. Nothing
  - b. Calls rendering pipeline
  - c. Creates 3D content
  - d. Computes scene lighting
The Rendering Pipeline

3D Content
- Needs to represent models for
  - Shapes (objects)
  - Relations between different shapes
  - Object materials
  - Light sources
  - Camera

Shapes: Representation options
- Volumetric - Boolean algebra with volumetric primitives
  - Spheres, cones, cylinders, tori, ...
- Boundary representation - union of surface patches
  - Single basic primitive - Triangle Mesh
  - Higher order surface/curve primitives

Shapes - Curves/Surfaces
- Mathematical representations:
  - Explicit functions
  - Parametric functions
  - Implicit functions

Shapes: Explicit Functions
- Curves:
  - \( y = \sin(x) \)
  - Only works in 2D

- Surfaces:
  - \( z = \sin(x) + \cos(y) \)
  - Cannot define arbitrary shapes in 3D

Shapes: Parametric Functions
- Curves:
  - 2D: \( x \) and \( y \) are functions of a parameter value \( t \)
  - 3D: \( x, y, \) and \( z \) are functions of a parameter value \( t \)

\[
C(t) = \begin{pmatrix}
\cos(t) \\
\sin(t) \\
t
\end{pmatrix}
\]
Computer Graphics

Shapes: Parametric Functions
- Surfaces:
  - Surface $S$ is defined as a function of parameter values $s, t$
  - Names of parameters can be different to match intuition:
    $$S(\phi, \theta) = \begin{pmatrix} \cos(\phi) \cos(\theta) \\ \sin(\phi) \cos(\theta) \\ \sin(\theta) \end{pmatrix}$$

Shapes: Implicit
- Surface (3D) or Curve (2D) defined by zero set (roots) of function
  - E.g:
    $$S(x, y, z) : x^2 + y^2 + z^2 - 1 = 0$$

Shapes: Triangle Meshes
- Triangle = 3 vertices

Open GL: (More) Shape Primitives
- `glPointSize(float size);`
- `glLineWidth(float width);`
- `glColor3f(float r, float g, float b);`
- `glBegin(GL_TRIANGLES);`  
  `glVertex3f(0.0f, 0.5f, 0.0f);`  
  `glVertex3f(-0.5f, -0.5f, 0.0f);`  
  `glVertex3f(0.5f, -0.5f, 0.0f);`  
  `glEnd();`

OpenGL – Shape Primitives
- How to interpret geometry
  - `glBegin(<mode of geometric primitives>);`
  - `mode = GL_TRIANGLE, GL_POLYGON, etc.`
- Feed vertices
  - `glVertex3f(-1.0, 0.0, -1.0);`
  - `glVertex3f(1.0, 0.0, -1.0);`
  - `glVertex3f(0.0, 1.0, -1.0);`
- Done
  - `glEnd();`

The Rendering Pipeline

Copyright  A. Sheffer, 2013, UBC
Modeling and Viewing Transformations

- Placing objects - Modeling transformations
  - Map points from object coordinate system to world coordinate system

- Placing camera - Viewing transformation
  - Map points from world coordinate system to camera (or eye) coordinate system

Modeling Transformations: Object Placement

Viewing Transformation: Camera Placement

Modeling & Viewing Transformations

- Types of transformations:
  - Rotations, scaling, shearing

- Translations

- Other transformations (not handled by rendering pipeline):
  - Freeform deformation

Modeling & Viewing Transformation

- Linear transformations
  - Rotations, scaling, shearing
  - Can be expressed as 3x3 matrix
  - E.g. scaling (non uniform):

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix} =
\begin{pmatrix}
  2 & 0 & 0 \\
  0 & 3 & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\]
Computer Graphics

Modeling & Viewing Transformation

- Affine transformations
  - Linear transformations + translations
  - Can be expressed as 3x3 matrix + 3 vector
  - E.g. scale + translation:

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix}
= \begin{pmatrix}
  2 & 0 & 0 \\
  0 & 3 & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
+ \begin{pmatrix}
  t_x \\
  t_y \\
  t_z
\end{pmatrix}
\]

- Another representation: 4x4 homogeneous matrix

Rendering Pipeline

Perspective Transformation

- Purpose:
  - Project 3D geometry to 2D image plane
  - Simulates a camera

- Camera model:
  - Pinhole camera (single view point)
  - More complex camera models exist, but are less common in CG
Perspective Projection

- In computer graphics:
  - Image plane conceptually in front of center of projection
  - Perspective transformations – subset of projective transformations
  - Linear & affine transformations also belong to this class
  - All projective transformations can be expressed as 4x4 matrix operations

Perspective Transformation

The Rendering Pipeline

- Removing invisible geometry
  - Geometry outside viewing frustum
  - Plus too far or too near one

Clipping

Scan Conversion/Rasterization

- Convert continuous 2D geometry to discrete
- Raster display – discrete grid of elements
- Terminology
  - Pixel: basic element on device
  - Resolution: number of rows & columns in device
    - Measured in
      - Absolute values (1K x 1K)
      - Density values (300 dots per inch)
- Screen Space: Discrete 2D Cartesian coordinate system of the screen pixels
Scan Conversion

Problem:
- Line is infinitely thin, but image has finite resolution
- Results in steps rather than a smooth line
  - Jaggies
  - Aliasing
- One of the fundamental problems in computer graphics

Color interpolation
- Linearly interpolate per-pixel color from vertex color values
- Treat every channel of RGB color separately

Example:
- red
- green
- blue
Texturing

- Issues:
  - Computing 3D/2D map (low distortion)
  - How to map pixel from texture (texels) to screen pixels
  - Texture can appear widely distorted in rendering
  - Magnification / minification of textures
  - Filtering of textures
  - Preventing aliasing (anti-aliasing)
The Rendering Pipeline

Geometric Content → Model/View Transform → Lighting → Perspective Transform → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Framebuffer

Computer Graphics

Rendering Pipeline/OpenGL

Without Hidden Line Removal

Hidden Line Removal

Hidden Surface Removal

Depth Test /Hidden Surface Removal

- Remove invisible geometry
- Parts that are hidden behind other geometry
- Possible Implementations:
  - Pixel level decision
  - Depth buffer
  - Object space decision
  - E.g. intersection order for ray tracing

Copyright A. Sheffer, 2013, UBC
Blending

- Blending:
  - Final image: specify pixel color
  - Draw from farthest to nearest
  - No blending - replace previous color
  - Blending: combine new & old values with some arithmetic operations
- Frame Buffer: video memory on graphics board that holds resulting image & used to display it

Clicker Quiz

- Which type of function is used in this curve description: 
  \[
  \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \sin \alpha \\ \cos \alpha \end{pmatrix}
  \]
  A. Implicit
  B. Explicit
  C. Parametric
  D. Quadratic

Not Handled: Reflection/Shadows

Copyright  A. Sheffer, 2013, UBC