Chapter 2
Basics of Computer Graphics:
Rendering Pipeline/OpenGL

Todo’s reminder
- Discussion Group: register
- Assignment 1
  - Test programming environment on lab computers/Set up programming environment on your laptop (optional)
  - Start assignment (have most background after class today)
- Reading (in Shirley: Introduction to CG)
  - Math refresher: Chapters 2, 4
    - Lots of math coming soon- be ready !!!
  - 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 patterns of operations
  - Rendering Pipeline
Offline rendering:
- Ray-tracing
- Global illumination

Copyright  A. Sheffer, 2011, UBC
Advantages of pipeline structure

- Logical separation of different components, modularity
- Easy to parallelize:
  - Earlier stages can already work on new data while later stages still work with previous data
  - Similar to pipelining in modern CPUs
  - But much more aggressive parallelization possible (special purpose hardware!)
  - Important for hardware implementations!
- Only local knowledge of the scene is necessary

Disadvantages:

- Limited flexibility
- Some algorithms would require different ordering of pipeline stages
- Hard to achieve while still preserving compatibility
- Only local knowledge of the scene is available – doesn’t support:
  - Shadows
  - Global illumination

Discussion

Rendering Pipeline Implementation: OpenGL/Glut

API for graphics hardware
- Started in 1989 by Kurt Akeley
- Designed to exploit GPU
- Implemented on many different platforms
- Low level, powerful flexible
- Pipeline processing
  - Communication via state setting
  - Event driven

OpenGL

- Set state once, remains until overwritten
  - glColor3f(1.0, 1.0, 0.0) -> set color to yellow
  - glSetClearColor(0.0, 0.0, 0.2) -> dark blue bg
  - glEnable(LIGHT0) -> turn on light
  - glEnable(GL_DEPTH_TEST) -> hidden surf.

(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 20
- Lighting Models (week 8)
- Texture mapping (week 9/10)
- Review & Midterm (week 10)
  - Midterm: Nov 10
- Ray Tracing (week 11)
- Shadows (week 11/12)
- Modeling (content creation) (week 12/13)
- Review (last lecture)
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 DrawWorld() {
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
glClear( GL_COLOR_BUFFER_BIT );
    glRotatef(angle,0,0,1);
    ...  // draw triangle
    glutSwapBuffers();
}
```

GLUT: OpenGL Utility Toolkit

- Event driven !!!

```c
int main(int argc, char **argv) {
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
glutInitWindowSize( 640, 480 );
glutCreateWindow( "openGLDemo" );
glutDisplayFunc( DrawWorld );
glutIdleFunc(Idle);
glClearColor( 1,1,1 );
glutMainLoop();
    return 0;       // never reached
}
```

GLUT Example

```c
void Idle() {
    angle += 0.05;
    glutPostRedisplay();
}
```

GLUT and GLU primitives

- glsSphere(...)
- glsCylinder(...)
- glsSolidSphere(GLdouble radius, GLint slices, GLint stacks)
- glsWireSphere(...)
- glsSolidCube(GLdouble size)
- glsWireCube(...)
- glsSolidTeapot(...)
- glsWireTorus(...)
- glsSolidTeapot(...)

Note:
- Have limited set of parameters
- Control via global transformations (see a1 template)
- Need to save/restore setting
GLUT and GLU primitives

- Example (from a1): draw ball at random positions

```c
// Setup transformation matrix for this ball.

glPushMatrix();
glLoadIdentity();

// Now have coordinate system with 0,0,0 at ball centre
glutSolidSphere(radius, 12, 4);

// Restore the transformation matrix.

glPopMatrix();
```

Basic Transformations:

```c
// Different basic transformations

glTranslatef();
glScalef();
glRotatef();
```

Depth buffer

- For visibility (hidden surface removal)
  - stores a z-value for every pixel
  - smaller z means “closer”

```c
// allocate depth buffer
glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

// enabling the depth test
glEnable( GL_DEPTH_TEST );

// clearing the depth buffer for each frame
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Open GL: (Some) Primitives

```c
// BEGIN GEOMETRY

GL_TRIANGLES

// VERTICES

glColor3f(0,1,0);
glVertex3f( 0.0f, 0.5f, 0.0f );
glVertex3f( 0.5f, 0.0f, 0.0f );
glVertex3f( -0.5f, 0.5f, 0.0f );

// END GEOMETRY

glEnd();
```

Assignment 1

- Experience OpenGL & GLUT
- Have FUN
- Deadline: Sep 23
Rendering Pipeline in (More) Detail

The Rendering Pipeline

Geometric Primitives

- Different philosophies:
  - Collections of complex shapes
    - Spheres, cones, cylinders, tori, ...
  - One simple type of geometric primitive
    - Triangles or triangle meshes
  - Small set of complex primitives with adjustable parameters
    - E.g. “all polynomials of degree 2”
    - Splines, NURBS (details in CPSC 424)
    - Implicits

Geometric Primitives

- Mathematical representations:
  - Explicit functions
  - Parametric functions
  - Implicit functions

(Not only) Geometric Content

- Needs to represent models for
  - Geometric primitives
  - Relations between different primitives (transformations)
  - Object materials
  - Light sources
  - Camera

Explicit Functions

- Curves:
  - $y$ is a function of $x$: $y := \sin(x)$
  - Only works in 2D

- Surfaces:
  - $z$ is a function of $x$ and $y$: $z := \sin(x) + \cos(y)$
  - Cannot define arbitrary shapes in 3D
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 \( t \)
  \[
  C(t) := \begin{pmatrix}
  \cos(t) \\
  \sin(t) \\
  t
  \end{pmatrix}
  \]

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

Geometric Content

- Implicit Surfaces:
  - Surface defined by zero set (roots) of function
  - E.g.:
  \[
  S(x, y, z) : x^2 + y^2 + z^2 - 1 = 0
  \]

Rendering Pipeline

- Triangles and Triangle Meshes:

The Rendering Pipeline

- 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 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. rotation:

\[
\begin{pmatrix}
  x' \\
y' \\
z'
\end{pmatrix} =
\begin{pmatrix}
  \cos(\phi) & -\sin(\phi) & 0 \\
  \sin(\phi) & \cos(\phi) & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
\]

Modeling & Viewing Transformation
- Affine transformations
  - Linear transformations + translations
  - Can be expressed as 3x3 matrix + 3 vector
  - E.g. rotation + translation:

\[
\begin{pmatrix}
  x' \\
y' \\
z'
\end{pmatrix} =
\begin{pmatrix}
  \cos(\phi) & -\sin(\phi) & 0 \\
  \sin(\phi) & \cos(\phi) & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y + t_y \\
z + t_z
\end{pmatrix}
\]

Another representation: 4x4 homogeneous matrix

The Rendering Pipeline
- Geometric Content
- Model/View Transformation
- Geometry Processing
- Lighting
- Perspective Transform.
- Clipping
- Scan Conversion
- Rasterization
- Texturing
- Depth Test
- Blending
- Fragment Processing
- Frame-buffer

Copyright A. Sheffer, 2011, UBC
Computer Graphics

Lighting

Rendering Pipeline/OpenGL

Perspective Transformation

- Purpose:
  - Project 3D geometry to 2D image plane
  - Simulates a camera
- Camera model:
  - Pinhole camera (single view point)
  - Other, more complex camera models also exist in computer graphics, but are less common
    - Thin lens cameras
    - Full simulation of lens geometry

Complex Lighting and Shading

Perspective Projection

- Pinhole Camera:
  - Light shining through a tiny hole into a dark room yields upside-down image on wall

The Rendering Pipeline

Perspective Transformation

- Pinhole Camera:
  - Light shining through a tiny hole into a dark room yields upside-down image on wall

Copyright  A. Sheffer, 2011, UBC
**Computer Graphics**

- **Pinhole Camera**

- **Perspective Transformation**
  - 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

---

**Rendering Pipeline/OpenGL**

- **The Rendering Pipeline**
  - Geometric Content
  - Model/View Transform.
  - Lighting
  - Perspective Transform.
  - Clipping
  - Scan Conversion
  - Texturing
  - Depth Test
  - Blending
  - Frame-buffer

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

---

Copyright  A. Sheffer, 2011, UBC
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

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

Copyright  A. Sheffer, 2011, UBC
Color interpolation

Example:

Scan Conversion

The Rendering Pipeline

Geometric Content

Model/View Transform

Lighting

Perspective Transform

Clipping

Geometric Content

Model/View Transform

Lighting

Perspective Transform

Clipping

Scan Conversion

Texturing

Depth Test

Blending

Frame-buffer

Texturing

Issues:

- 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 ➔ Geometry Processing

Scan Conversion ➔ Texturing ➔ Depth Test ➔ Blending ➔ Frame buffer

The Rendering Pipeline

Hidden Surface Removal

Without Hidden Line Removal

Hidden Line Removal

Depth Test /Hidden Surface Removal

- Remove invisible geometry
  - Parts that are hidden behind other geometry
- Possible Implementations:
  - Per-fragment decision
    - Depth buffer
  - Object space decision
    - Clipping polygons against each other
    - Sorting polygons by distance from camera
The Rendering Pipeline

- Geometric Content
- Model/View Transform
- Lighting
- Perspective Transform
- Clipping
- Scan Conversion
- Texturing
- Depth Test
- Blending
- Rasterization
- Fragment Processing
- Frame-buffer

Blending
- Blending:
  - Final image: write fragments to pixels
  - 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

Not Handled: Reflection/Shadows

Copyright  A. Sheffer, 2011, UBC