The OpenGL Pipeline

CS 148, Summer 2012
Introduction to Computer Graphics and Imaging
Justin Solomon
The OpenGL Pipeline

CS 148, Summer 2012
Introduction to Computer Graphics and Imaging
Justin Solomon
<clarification>
Nonlinearity in Rasterized Depths

\[ z_{\text{world}} \rightarrow z = z_{\text{near}} + z_{\text{far}} - \frac{z_{\text{near}} z_{\text{far}}}{z_{\text{world}}} \]

\[ \frac{dz}{d z_{\text{world}}} = \frac{z_{\text{near}} z_{\text{far}}}{z_{\text{world}}^2} \]

\[ \Delta z \approx \frac{z_{\text{near}} z_{\text{far}} \Delta z_{\text{world}}}{z_{\text{world}}^2} \]
Nonlinearity in Rasterized Depths

Better resolution when closer to the screen.

\[ \Delta z \approx \frac{z_{\text{near}} z_{\text{far}} \Delta z_{\text{world}}}{z_{\text{world}}^2} \]

\[ \Delta z_{\text{world}} \approx \frac{z_{\text{world}}^2 \Delta z}{z_{\text{near}} z_{\text{far}}} \]
Theory of rasterization
Advice

Leave implementation of low-level features to the experts.
Why?

- Abstract away hardware differences
- Rasterization should be fast
Introducing...

Industry standard API for computer graphics
Not the Only One

Microsoft® DirectX®

RenderMan

3dfx

Glide API
Not the Only One

For interactive applications; not specific to hardware or OS
Related APIs in the OpenGL Family

- WebGL
- OpenGL ES
- OpenGL SC
- WebCL
- OpenVG
- OpenCL
Related APIs in the OpenGL Family
Related APIs in the OpenGL Family
Related APIs in the OpenGL Family

- WebGL
- OpenGL ES
- OpenGL SC
- WebCL
- OpenVG
- OpenCL
WebGL
WebGL
(Simplified) OpenGL Pipeline

Display List

Evaluator

Per-Vertex Operations
Primitive Assembly

Rasterization

Per-Fragment Operation

Frame Buffer

Texture Memory

Pixel Operations
(Unsimplified) OpenGL Pipeline
(Unsimplified) OpenGL Pipeline
Pieces of the Pipeline

Stores “subroutines”
Pieces of the Pipeline

Stores “subroutines”

Faster!
- Pre-compiled
- Store on GPU
- Pre-compute transformations
Pieces of the Pipeline

Stores “subroutines”

Faster!
- Pre-compiled
- Store on GPU
- Pre-compute transformations…sometimes
Pieces of the Pipeline

Construct geometric objects
Pieces of the Pipeline

Change meshed geometry

Store primitive shapes
Pieces of the Pipeline

Change meshed geometry

Store primitive shapes

Includes clipping!
Primitive [prim-i-tiv]:
A small piece of geometry that can be rendered; in OpenGL, triangles, quads, lines, points (usually).
Pieces of the Pipeline
Fragment [frag-muhnt]: The data necessary to generate a single pixel’s worth of a primitive.

http://en.wikipedia.org/wiki/Fragment_%28computer_graphics%29
Modify and combine per-pixel information
Pieces of the Pipeline

Prepare image to be displayed
OUT OF DATE.
OUT OF DATE.

cf. next lecture!
OpenGL is a state machine

<table>
<thead>
<tr>
<th>OpenGL Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>glColor3f(...)</td>
</tr>
<tr>
<td>glEnable(...)</td>
</tr>
<tr>
<td>glLineStipple(...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OpenGL Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>glGetFloatv(...)</td>
</tr>
<tr>
<td>glEnable (...)</td>
</tr>
<tr>
<td>glIsEnabled (...)</td>
</tr>
<tr>
<td>glGetLineStipple (...)</td>
</tr>
</tbody>
</table>
What OpenGL Is Not

A windowing system

Platform independent!
GLUT

- Simple cross-platform windowing API
- Bindings: C, C++, Fortran, Ada, ...
- Features:
  - Multiple windows, menus
  - Keyboard/mouse/other input
  - Assorted callbacks: idle, timers
  - Basic font support
  - `glutSolidTeapot, glutSolidSphere, glutSolidCube, ...`

GL Utility Toolkit
GLUT

- Simple cross-platform windowing API
- Bindings: C, C++, Fortran, Ada, ...
- Features:
  - Multiple windows, menus
  - Keyboard/mouse/other input
  - Assorted callbacks: idle, timers
  - Basic font support
  - `glutSolidTeapot`, `glutSolidSphere`, `glutSolidCube`, ...

GL Utility Toolkit
Simple cross-platform windowing API

Bindings: C, C++, Fortran, Ada, ...

Features:

- Multiple windows, menus
- Keyboard/mouse, other input
- Assorted callbacks: idle, timers
- Basic font support

`glutSolidTeapot, glutSolidSphere, glutSolidCube, ...`

GL Utility Toolkit
GL Utility Library

- High-level graphics commands
- Not included in OpenGL ES
- Some interesting features:
  - Mapping between world and screen coordinates
  - Texturing support
  - Tessellation and other geometric utilities
  - OpenGL error code lookup
  - More primitives: spheres, cylinders, disks, ...
  - Camera support: gluLookAt, gluOrtho2D, ...
Adding Geometry to the Scene

```c
glBegin(GL_TRIANGLES);
glVertex3d(0, 0, 0);
glVertex3d(1, 0, 0);
glVertex3d(0, 1, 0);
glEnd(/*GL_TRIANGLES*/);
```

Option 1: Immediate mode
Adding Geometry to the Scene

Option 1: Immediate mode

```c
glBegin(GL_TRIANGLES);
  glVertex3d(0, 0, 0);
  glVertex3d(1, 0, 0);
  glVertex3d(0, 1, 0);
glEnd(/*GL_TRIANGLES*/);
```
Adding Geometry to the Scene

```
glBegin(GL_TRIANGLES);
glVertex3d(0, 0, 0);
glVertex3d(1, 0, 0);
glVertex3d(0, 1, 0);
glEnd(/*GL_TRIANGLES*/);
```

Not needed

Option 1: Immediate mode
Adding Geometry to the Scene

```c
glBegin(GL_TRIANGLES);
glVertex3d(0, 0, 0);
glVertex3d(1, 0, 0);
glVertex3d(0, 1, 0);
glVertex3d(1, 0, 0);
glVertex3d(1, 1, 0);
glVertex3d(0, 1, 0);
glEnd(/*GL_TRIANGLES*/);
```

Option 1: Immediate mode
Adding Geometry to the Scene

```c
glBegin(GL_TRIANGLE_STRIP);
    glVertex3d(0, 0, 0);
    glVertex3d(1, 0, 0);
    glVertex3d(0, 1, 0);
    glVertex3d(1, 1, 0);
glEnd(/*GL_TRIANGLE_STRIP*/);
```

Option 1: Immediate mode
Adding Geometry to the Scene

```c
glBegin(GL_TRIANGLE_STRIP);
    glVertex3d(0, 0, 0);
    glVertex3d(1, 0, 0);
    glVertex3d(0, 1, 0);
    glVertex3d(1, 1, 0);
 glEnd(); /*GL_TRIANGLE_STRIP*/;
```

Option 1: Immediate mode
GLuint boxList;
boxList = glGenLists(1);
glNewList(boxList, GL_COMPILE);
    // draw box
    glEndList(boxList);
...
glCallList(boxList);
Option 3: Vertex buffer

GLfloat vertices[] = {...};

glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glDrawArrays(GL_TRIANGLES, 0, 36);
glDisableClientState(GL_VERTEX_ARRAY);
Recall: Backface Culling

None  
Backface culling  
Hidden surface removal

Cull triangles with back-facing normals

http://medialab.di.unipi.it/web/UM/Waterloo/node70.html
Specifying Triangle Orientation

Default: `glFrontFace(GL_CCW)`
Specifying Triangle Orientation

Default: `glFrontFace(GL_CCW)`

Right-handed!
Specifying Triangle Orientation

```
glFrontFace(GL_CW)
```
Specifying Triangle Orientation

```cpp
glDisable(GL_CULL_FACE)
```
Specifying Triangle Orientation

glCullFace(GL_FRONT_AND_BACK)
Specifying Triangle Orientation

`glFrontFace(GL_CCW)`
`glCullFace(GL_FRONT)`
Computing Face Normals

\[
\vec{N} = \frac{(\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1)}{\| (\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1) \|}
\]
Aside: Storing Meshes

xyz/xyz/xyz
xyz/xyz/xyz
xyz/xyz/xyz
...

Low level: “Triangle soup”
Aside: Storing Meshes

Likely to repeat

xyz/xyz/xyz
xyz/xyz/xyz
xyz/xyz/xyz
...

Low level: “Triangle soup”
Aside: Storing Meshes

Medium level: Triangle/vertex lists

Floats

xyz

... 

Integers

v_1 v_2 v_3

v_1 v_2 v_3

...
Aside: Storing Meshes

Medium level: Triangle/vertex lists

Hard query: “Neighbors of vertex v”
Aside: Storing Meshes

High level: Halfedge
Shading Options

glColor3f
Per-Vertex Shading

Barycentric coordinates!
Shading Options

glLineStipple

glPolygonStipple
Shading Options

\texttt{glHint(GL\_LINE\_SMOOTH\_HINT, GL\_NICEST)}
Shading Options

```c
glHint(GL_LINE_SMOOTH_HINT, GL_NICEST)
```
Shading Options

\texttt{glFogi(GL\_EXP,GL\_FOG\_START)}

\textit{Attenuates using depth buffer}
Shading Options

Stencil buffer

http://ofps.oreilly.com/titles/9780596804824/chblending.html
void glLoadMatrixd(const GLdouble *m)

- Loads a 4x4 projective transformation matrix
- Stored in *column-major* format
Composing Transformations

```c
glLoadMatrixf(A);
glMultMatrixf(B);
glMultMatrixf(C);
```

$$\vec{v}_{new} = A \cdot B \cdot C \cdot \vec{v}$$
Composing Transformations

```c
glLoadIdentityf(A); // translate right
glMultMatrixf(B); // rotate 45°
```

\[
\vec{u}_{new} = A \cdot B \cdot \vec{v}
\]
Convenience Functions

- `glTranslatef(tx, ty, tz)`
- `glRotatef(degrees, x, y, z)`
- `glScalef(sx, sy, sz)`
Hierarchical Modeling

Body
  Torso
  Head
  Shoulder
    LeftArm
      UpperArm
      LowerArm
    Hand
    RightArm
      UpperArm
      LowerArm
    Hand
  Hips
    LeftLeg
      UpperLeg
      LowerLeg
    Foot
    RightLeg
      UpperLeg
      LowerLeg
    Foot

Work in local coordinate system
Hierarchical Modeling

Repeated elements!
Hierarchical Modeling

```plaintext
glTranslatef(0,1.5,0);
drawTorso();

```
OpenGL Transformation Model

\[ T = M_{projection} \cdot M_{camera} \cdot M_{model} \]

- Camera coordinates to normalized device coordinates
- Local coordinates to camera coordinates
OpenGL Transformation Model

\[ T = M_{\text{projection}} \cdot M_{\text{camera}} \cdot M_{\text{model}} \]

- Camera coordinates to normalized device coordinates
- Local coordinates to camera coordinates

\begin{align*}
\text{GL}_{\_}\text{PROJECTION} & : \quad \text{glMatrixMode(GL}_{\_}\text{PROJECTION}); \\
\text{GL}_{\_}\text{MODELVIEW} & : \quad \text{glMatrixMode(GL}_{\_}\text{MODELVIEW}); \\
\end{align*}

Applied to all geometry
OpenGL Transformation Model

\[ T = M_{\text{projection}} \cdot M_{\text{camera}} \cdot M_{\text{model}} \]

Camera coordinates to
normalized device
coordinates

GL_PROJECTION

Camera coordinates
coordinates

GL_MODELVIEW

\[ \text{glMatrixMode(GL_PROJECTION);} \]
\[ \text{glMatrixMode(GL_MODELVIEW);} \]

Can do whatever you
want with these.

\text{Applied to all geometry}
Normalized Device Coordinates

Image Plane

$[-1, 1]^3$
Transformation Example

http://www.sulaco.co.za/pushing_and_popping_the_matrix_stack_tutorial.htm
procedure DrawTriangleFan(CirclePoints: GLint; scale : GLfloat );
var i : GLint;
  Angle : GLfloat;
begin
  glBegin(GL_TRIANGLE_FAN);
  for i := 0 to CirclePoints - 1 do
  begin
    Angle := 2*PI*i/CirclePoints;
    glVertex2f(cos(angle)*scale, sin(angle)*scale);
  end;
  glEnd();
end;
procedure DrawWheel();
begin
  DrawTriangleFan(20,0.5);
end;

procedure DrawBolt();
begin
  DrawTriangleFan(6,0.05);
end;

http://www.sulaco.co.za/pushing_and_popping_the_matrix_stackTutorial.htm
procedure DrawWheelAndBolts(RightSide : boolean);
var i : GLint;
begin
    glColor3f(0.5,0.5,0.5);
    DrawWheel();
    for i := 0 to 4 do
        begin
            glPushMatrix();
            glRotatef(72.0*i,0.0,0.0,1.0);
            if RightSide then
                glTranslatef(0.2,0.0,0.01)
            else
                glTranslatef(0.2,0.0,-0.01);
            glColor3f(0.3,0.3,0.3);
            DrawBolt();
            glPopMatrix();
        end;
end;
procedure DrawBodyAndWheelAndBolts();
begin
    glColor3f(1.0,0.0,0.0);
    DrawCarBody();
    glPushMatrix();
        glTranslatef(-1.0,-0.5,1.01);  DrawWheelAndBolts(True);
    glPopMatrix();
    glPushMatrix();
        glTranslatef(-1.0,-0.5,-1.01);  DrawWheelAndBolts(False);
    glPopMatrix();
    glPushMatrix();
        glTranslatef(1.0,-0.5,-1.01);  DrawWheelAndBolts(False);
    glPopMatrix();
    glPushMatrix();
        glTranslatef(1.0,-0.5,1.01);  DrawWheelAndBolts(True);
    glPopMatrix();
end;
The OpenGL Pipeline

CS 148, Summer 2012
Introduction to Computer Graphics and Imaging
Justin Solomon