Introduction to OpenGL

Estelle Duveau - estelle.duveau@inria.fr

MoSIG1, Introduction to Computer Graphics, 25/02/2009

Planning

- **Lecture**: introduction to OpenGL
  Lab: first steps in OpenGL and modeling - 25/02/2009
- **Lecture/Lab**: transformations and hierarchical modeling - 04/03/2009
- **Lecture**: lights and materials in OpenGL - 11/03/2009
- **Lab**: lights and materials in OpenGL - 18/03/2009
- **Lecture**: textures in OpenGL
  Lab: textures in OpenGL - 25/03/2009
- **Lab**: procedural animation - 01/04/2009
- **Lab**: physical animation: particle systems - 08/04/2009
- **Lab**: physical animation: collisions - 22/04/2009

References

D. Shreiner, M. Woo, J. Neider, T. Davis
OpenGL Programming Guide
aka the red book
http://opengl-redbook.com

What is it?

- API (Application Programming Interface) for graphics hardware
- Non-dependant on the architecture or programming language
- Developed in 1989 (GL) by Silicon Graphics, extended to other architectures in 1993 (OpenGL)
- About 250 commands
**It's a state machine!**

State machine = each parameter retains its value and is used with that value until being explicitly changed

Parameters can be:

- **modes**: shading mode, matrix manipulated ...
  
  ```
  glEnable(GL_LIGHT0); glDisable(GL_LIGHT1);
  ```

- **booleans**: lights on/off, blend colors, ...
  
  ```
  glShadeMode(GL_FLAT); glShadeMode(GL_SMOOTH);
  ```

- **scalar values**: colors, viewpoint, ...
  
  ```
  glColor3f(0.0,0.0,0.0); glNormal3f(1.0,0.0,0.0);
  ```

---

**Things it can NOT do**

- Can NOT create nor manage a viewer
- Can NOT manage complex objects: only 3 types of geometric primitives (points, lines, polygons)

- Additional libraries needed:
  - **GLU**: openGL Utility library: more complex 3D models
  - **GLUT**: openGL Utility Toolkit: viewer

---

**Example: OpenGL with GLUT**

```c
int main(int argc, char** argv){
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(250,250);
    glutInitWindowPosition(100,100);
    glutCreateWindow("hello");
    init(); // to define with OpenGL
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutKeyboardFunc(keyboard);
    glutMainLoop(); // Let's go! Scene displayed
    return 0;
}
```

---

**Plan**

1. **Introduction**
2. **Pipeline**
   - Graphics pipeline
   - OpenGL pipeline
   - OpenGL syntax
3. **Modeling**
   - Procedural modeling
   - OpenGL primitives
   - Arrays
4. **Conclusion**

---

**Graphics Pipeline**

1. Create 3D models (modeling)
2. Build the scene from instances of models placed in a world frame (modeling transformation)
3. Convert to camera frame (culling, frustrum)
4. Convert to screen frame (projection)
5. Compute image (rasterization)
Plan

1 Introduction
2 Pipeline
   Graphics pipeline
   OpenGL pipeline
   OpenGL syntax
3 Modeling
   Procedural modeling
   OpenGL primitives
   Arrays
4 Conclusion

Introduction to OpenGL

Introduction
Pipeline
Graphics pipeline
OpenGL pipeline
OpenGL syntax
Modeling
Procedural modeling
OpenGL primitives
Arrays
Conclusion

Plan

1 Introduction
2 Pipeline
   Graphics pipeline
   OpenGL pipeline
   OpenGL syntax
3 Modeling
   Procedural modeling
   OpenGL primitives
   Arrays
4 Conclusion

OpenGL Pipeline

Create OpenGL context
Loop:
1 Manage mouse/keyboard events
2 Display
   1 Clear screen
   2 Viewpoint
   3 For each object:
      1 Place object
      2 Modify state machine
      3 Draw

Basic example - 1/2

Display:
void display () {
1 Clear screen
   glClear(GL_COLOR_BUFFER_BIT);
2 Viewpoint
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
3 For each object:
   1 Place object
      glMatrixMode(GL_MODELVIEW);
   2 Modify state machine
      glColor3f(1.0,1.0,1.0);
   3 Draw
      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(); // Execute OpenGL commands in hold
}

Basic example - 2/2

void display () {
   glClear(GL_COLOR_BUFFER_BIT);
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,0.1,0.0,1.0,-1.0,1.0); 
   glMatrixMode(GL_MODELVIEW);
   glColor3f(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();
}
OpenGL Syntax - 1/2

Reminder:

- modes: `gl[MODE]Mode(GL_VALUE)`
- booleans: `glEnable(GL_VALUE)`

⇒ OpenGL constants start with `GL_`

- scalar values:
  
  ```
  glColor3f(1.0,1.0,1.0);
  
  glColor3fv(color_array);
  
  The argument is a vector (or array) of 3 floats
  (GLfloat color_array[] = { 1.0,0.0,0.0 } ; )
  ```

OpenGL Syntax - 2/2

OpenGL suffixes and types

<table>
<thead>
<tr>
<th>Type</th>
<th>GL Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b</code></td>
<td>integer (8 bits)</td>
</tr>
<tr>
<td><code>s</code></td>
<td>integer (16 bits)</td>
</tr>
<tr>
<td><code>i</code></td>
<td>integer (32 bits)</td>
</tr>
<tr>
<td><code>f</code></td>
<td>real (32 bits)</td>
</tr>
<tr>
<td><code>d</code></td>
<td>real (64 bits)</td>
</tr>
<tr>
<td><code>ub</code></td>
<td>unsigned integer (8 bits)</td>
</tr>
<tr>
<td><code>us</code></td>
<td>unsigned integer (16 bits)</td>
</tr>
<tr>
<td><code>ul</code></td>
<td>unsigned integer (32 bits)</td>
</tr>
</tbody>
</table>

Plan

1. Introduction
2. Pipeline
   - Graphics pipeline
   - OpenGL pipeline
   - OpenGL syntax
3. Modeling
   - Procedural modeling
   - OpenGL primitives
   - Arrays
4. Conclusion

Plan

1. Introduction
2. Pipeline
   - Graphics pipeline
   - OpenGL pipeline
   - OpenGL syntax
3. Modeling
   - Procedural modeling
   - OpenGL primitives
   - Arrays
4. Conclusion

Procedural modeling

Complex object = combination of elementary elements:

- Points (vertices): coordinates in a given reference frame
- Lines: segments
- Polygons: simple convex polygons

Example - planar pentagon

```
glBegin(GL_POLYGON);
glVertex2f(0.0, 0.0);
glVertex2f(0.0, 3.0);
glVertex2f(4.0, 3.0);
glVertex2f(6.0, 1.5);
glVertex2f(4.0, 0.0);
glEnd();
glFlush();
```
Introduction to OpenGL

1. Introduction
2. Pipeline
   - Graphics pipeline
   - OpenGL pipeline
   - OpenGL syntax
3. Modeling
   - Procedural modeling
   - OpenGL primitives
   - Arrays
4. Conclusion

Parameters

- Point size (in pixels): `glPointSize(2.0);`
- Line width (in pixels): `glLineWidth(3.0);`
- Line drawing: many stippling styles
- Different renderings for front and back faces:
  - `glPolygonMode(GL_FRONT, GL_FILL);`
  - `glPolygonMode(GL_BACK, GL_LINE);`
- Culling: `glCullFace(GL_BACK);`: back-faces non-visible
- The color, normal, ..., at each vertex can be specified
- ...
- Get current values: `glGetFloatv(GL_LINE_WIDTH);`

Example: normals

```c
glBegin(GL_POLYGON);
glNormal3fv(n0);
glVertex3fv(v0);
glNormal3fv(n1);
glVertex3fv(v1);
glNormal3fv(n2);
glVertex3fv(v2);
glEnd();
```

Beware of the order: parameter (i.e. normal) before coordinates

Arrays

- An array for each type of data: coordinates, normal, color, texture, ...
- Create an array: `glEnableClientState(GL_NORMAL_ARRAY);`
- Use that array:
  ```c
  glColorPointer(3, GL_FLOAT, 5*sizeof(GLfloat), color);
  ```
  → `stride` = offset (in bytes) between 2 consecutive data elements
- Access an element: `glArrayElement(0);`: done in all active arrays
- Access several elements:
  ```c
  glDrawElements(GL_POLYGON, 5, GL_UNSIGNED_INT, vertices);
  ```
  Also `glMultiDrawElements(...)`, `glDrawRangeElements(...)`, ...
Plan

1 Introduction

2 Pipeline
   Graphics pipeline
   OpenGL pipeline
   OpenGL syntax

3 Modeling
   Procedural modeling
   OpenGL primitives
   Arrays

4 Conclusion

Introduction to OpenGL

Introduction
   Pipeline
   Graphics pipeline
   OpenGL pipeline
   OpenGL syntax

Modeling
   Procedural modeling
   OpenGL primitives
   Arrays

Conclusion

- Done:
  - General process
  - Modeling: geometric primitives

- Highlights:
  - State machine
  - Primitives
  - the redbook

- To do:
  - lab
  - modeling complex objects with primitives...next week!