Introduction to OpenGL

“3D Geometry”

Reading: Angel Ch.4 and Woo Ch.2
Points and Vectors

Points/Vectors are basic primitives for geometric representation

OpenGL does not define point/vector objects

In C we can define types for points:

```c
typedef GLfloat point3[3]; /* array of 3 floats */
typedef GLfloat vector3[3];
```

We can then instantiate a point or vector as:

```c
point3 p={1,2,3};
vector3 v={1,3,5};
```

Functions can be used to define operations for each type:

```c
w = add(u,v);    /* vectors u,v,w */
w = dot(u,v);
w = subtract(p1,p2); /* vector w & points p1,p2 */
```

In an object oriented language such as C++ point/vectors can be classes with a set of operations (add/dot product,...)
Example - Representing a Cube

typedef GLfloat point3[3];

point3 cube_vertices[8] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
{1.0,1.0,-1.0}, {-1.0,1.0,-1.0},
{-1.0,-1.0,1.0},{1.0,-1.0,0.0},
{1.0,1.0,1.0},{-1.0,1.0,1.0}};

/* define a quadrilateral for each side of cube */

glBegin(GL_POLYGON);                     /* 1st face */
   glVertex3fv(cube_vertices[0]);
   glVertex3fv(cube_vertices[3]);
   glVertex3fv(cube_vertices[2]);
   glVertex3fv(cube_vertices[1]);
glEnd();

.... remaining 5 faces
Example- Cube vertex data
Vertex Arrays

Using repeated OpenGL function calls to define polygonal objects is inefficient
~ 30 calls for one cube

**Vertex Arrays** encapsulate data for vertices polyhedral objects
6 types: vertex, color, normal, texture_coordinate, color_index, edge_flag
Each type contains an array of data for the vertices

Functions to use vertex arrays:

1. Enable type of array
   ```
   glEnableClientState(type-of-array);
   ```

2. Identify array of vertex data
   ```
   glVertexPoint(nvertex,array-data-type,step, pointer-to-array);
   ```

3. Render array of faces
   ```
   glDrawElements(face-type, nface,index-type,pointer-to-index);
   ```
Using Vertex Arrays

(1) **Enable functionality of vertex arrays**
    ```
    glEnableClientState(GL_VERTEX_ARRAY); /* vertices */
    glEnableClientState(GL_COLOR_ARRAY); /* vertex colours*/
    ```

(2) **Setup arrays** of vertices
    ```
    point3 vertices[8] = {...};
    point3 colors[8] = {...};
    ```

(3) **Identify arrays** are by passing a pointer to the array
    ```
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(3, GL_FLOAT, 0, colors);
    ```

(4) **Define faces** by indicies refering to the faces
    ```
    Glubyte cubeIndices[24] = {0, 3, 2, 1, ...., 0, 1, 5, 4}
    ```

(5) **Render**
    ```
    for (i=0, i<6; i++)
    glDrawElements(GL_POLYGON, 4, GL_UNSIGNED_BYTE, &cubeIndices[4*i]);
    or
    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);
    ```
Example - Drawing a Cube

typedef GLfloat point3[3];

point3 vertices[8] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
                      {1.0,1.0,-1.0}, {-1.0,1.0,-1.0},
                      {-1.0,-1.0,1.0},{1.0,-1.0,0.0},
                      {1.0,1.0,1.0},{-1.0,1.0,1.0}};

point3 color[8] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
                    {1.0,1.0,0.0}, {0.0,1.0,0.0},
                    {0.0,0.0,1.0},{1.0,0.0,1.0},
                    {1.0,1.0,1.0},{0.0,1.0,1.0}};

Glubyte cubeIndices[24]={0,3,2,1, 2,3,7,6, 0,4,7,3,
                         1,2,6,5, 4,5,6,7, 0,1,5,4};

glVertexPointer(3,GL_FLOAT,0,vertices);
glColorPointer(3,GL_FLOAT,0,color);

glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);
Frames in OpenGL

2 frames: camera frame and world frame

Model-view matrix positions world frame relative to camera
- 4x4 homogenous matrix
- part of OpenGL state
- set model view matrix by:
  (1) set current state to Model-view
      \texttt{glMatrixMode(GL_MODELVIEW)}
  (2) set current model view to identity
      \texttt{glLoadIdentity()}
  (3) transform model view
      \texttt{glMultMatrixf(pointer\_to\_matrix)} \quad \text{general transform}
      \texttt{glRotatef(45.0,0.0,0.0,1.0)} \quad \text{rotate 45deg about axis [001]}
      \texttt{glTranslatef(1.0,2.0,3.0)} \quad \text{translate by vector [123]}
      \texttt{glScalef(sx,sy,sz)} \quad \text{scale}
Transformations

**Rule:** the most recent transformation is applied first

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0,2.0,3.0);
glRotatef(45.0,0.0,0.0,1.0);
glTranslatef(-1.0,-2.0,-3.0);
```

Transform $T = \text{Identity} * \text{Translation}(1,2,3) * \text{Rotation}(45,001) * \text{Translation}(-1-2-3)$

Now for a point $P$ in world coordinates the resulting position $Q$ after the model view transformation above is:

$$Q = T \cdot P$$

where $T$ is a 4x4 homogenous transformation matrix
Example - Spinning the Cube
OpenGL provides matrix stacks which allow us to switch between transformations - up to 32 model view matrices (maybe more)

```cpp
glPushMatrix();  // put the current model view matrix on the stack
glPopMatrix();  // pops the most recent model view matrix off the stack
```

For example to perform a transformation and then return to the previous
```cpp
glMatrixMode(GL_MODELVIEW);

glPushMatrix();

/* setup new model view */
glLoadIdentity();
glTranslatef(1.0,2.0,3.0);
glRotatef(45.0,0.0,0.0,1.0);
glTranslatef(-1.0,-2.0,-3.0);

glPopMatrix();
```
Display Lists

Display lists allow us to define an object once and to use it many times.
  • The object is defined once and put in a display list
  • Object is redisplayed by a single call

Functions:
  glNewList(index, type) - create list
    ‘type’ GL_COMPILE - send to server only
    GL_COMPILE_EXECUTE send & display
  glEndList() - end list
  glCallList(index) - draw the list on the server using current state

  base= glGenLists(nlist) - create ‘nlist’ consecutive lists
    which start at base list
  glListBase(base) - set offset to base list

  glCallLists(nlist_display, array_data_type, array_pointer)
    - display lists referred to by array_pointer
Example - Display List for a Cube

#define CUBE 1  /* number for cube display list */

... define cube vertex + face ....

glNewList(CUBE,GL_COMPILE); /* create and send list */
    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);
    glEndList();

glCallList(CUBE);  /* display cube according to current state*/

... change state (transform/projection/color)....  
    glCallList(CUBE);  /* display another cube */
Summary

• Vertex Arrays used to specify lists of points

• Transformation applied to OpenGL ‘ModelView’ matrix (in reverse order: last transform applied first)

• OpenGL supports a stack of upto 32 model view matrices

• Display lists used to efficiently create multiple instances of an object