Display Lists

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

```c
void OsuSphere( float radius, int slices, int stacks )
{
    struct point top, bot;          // top, bottom points
    struct point *p;
    NumLngs = slices;
    NumLats = stacks;
    Pts = new struct point[ NumLngs * NumLats ];
    for( int ilat = 0; ilat < NumLats; ilat++ )
    {
        float lat = -M_PI/2.  +  M_PI * (float)ilat / (float)(NumLats-1);
        float xz = cos( lat );
        float y   = sin( lat );
        for( int ilng = 0; ilng < NumLngs; ilng++ )
        {
            float lng = -M_PI  +  2. * M_PI * (float)ilng / (float)(NumLngs-1);
            float x =  xz * cos( lng );
            float z = -xz * sin( lng );
            p = PtsPointer( ilat, ilng );
            p->x  = radius * x;
            p->y  = radius * y;
            p->z  = radius * z;
            p->nx = x;
            p->ny = y;
            p->nz = z;
            p->s = ( lng + M_PI    ) / ( 2.*M_PI );
            p->t = ( lat + M_PI/2. ) / M_PI;
        }
    }
}
```

Drawing a Sphere – Notice a lot of time-consuming Trig Function Calls!

Even worse, the trig calls are inside single or double-nested for-loops!

```c
top.x =  0.;          top.y = radius;        top.z = 0.;
top.nx = 0.;           top.ny = 1.;               top.nz = 0.;
top.s = 0.;           top.t    = 1.;
bot.x =  0.;           bot.y = -radius;       bot.z = 0.;
bot.nx = 0.;            bot.ny = -1.;              bot.nz = 0.;
bot.s = 0.;            bot.t    =  0.;
glBegin( GL_QUADS );
for( int ilng = 0; ilng < NumLngs-1; ilng++ )
{
    p = PtsPointer( NumLats-1, ilng );
    DrawPoint( p );
    p = PtsPointer( NumLats-2, ilng );
    DrawPoint( p );
    p = PtsPointer( NumLats-2, ilng+1 );
    DrawPoint( p );
    p = PtsPointer( NumLats-1, ilng+1 );
    DrawPoint( p );
}
glEnd( );
glBegin( GL_QUADS );
for( int ilng = 0; ilng < NumLngs-1; ilng++ )
{
    p = PtsPointer( 0, ilng );
    DrawPoint( p );
    p = PtsPointer( 0, ilng+1 );
    DrawPoint( p );
    p = PtsPointer( 1, ilng+1 );
    DrawPoint( p );
    p = PtsPointer( 1, ilng );
    DrawPoint( p );
}
glEnd( );
glBegin( GL_QUADS );
for( int ilat = 2; ilat < NumLats-1; ilat++ )
{
    for( int ilng = 0; ilng < NumLngs-1; ilng++ )
    {
        p = PtsPointer( ilat-1, ilng );
        DrawPoint( p );
        p = PtsPointer( ilat-1, ilng+1 );
        DrawPoint( p );
        p = PtsPointer( ilat, ilng+1 );
        DrawPoint( p );
        p = PtsPointer( ilat, ilng );
        DrawPoint( p );
    }
}
glEnd( );
```

You don't want to execute all that code every time you want to redraw the scene, so draw it once, store the numbers in GPU memory, and call them back up later

### Without a Display List:
The CPU re-computes and transmits the sphere coordinates every time they need to be drawn.

### With a Display List:
The CPU computes the coordinates once and then they are grabbed from GPU memory every time they need to be drawn.

You don't want to execute all that code every time you want to redraw the scene, so draw it once, store the numbers in GPU memory, and call them back up later

The solution is to incur the sphere-creation overhead once, and whenever the sphere needs to be re-drawn, just draw the saved numbers, not the equations.  This is a **Display List**.

#### Creating the Display List in `InitLists`:
1. How many unique, unused, consecutive DL identifiers to give back to you
2. The ID of the first DL in the unique, unused list
3. Opening a display list in (GPU) memory
4. All done with storing the numbers in the DL

#### Calling up the Display List in `Display`:
1. The coordinates, etc. end up in memory instead of being sent to the display
2. Pull all the coordinates, etc. from memory, just as if the code to generate them had been executed here

---

```
// a global GLuint variable:
SphereList = glGenLists( 1 );
glNewList( SphereList, GL_COMPILE );
OsuSphere( 5., 30, 30 );
glEndList( );
gCallList( SphereList );
```

---
A Common Display List Misconception

Let's say that we are creating a rectangle in a Display List, like this:

```c
float L, W;  // length and width global variables
int RectList; // rectangle display list
L = 10.; W = 5.;
gNewList( RectList, GL_COMPILE );
<< draw a rectangle using L and W >>
gEndList();
```

Then, when we go to use the DL, we do this:

```c
L = 4.; W = 2.;
gCallList( RectList );
```

What size rectangle will it draw? 10x5? 4x2?

It will draw a 10x5 rectangle. Display Lists bake in the numbers. They retain no knowledge of what variables were used to create those numbers!