Display Lists

Mike Bailey
mjb@cs.oregonstate.edu

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

void Sphere( 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;
        }
    }
}

void Sphere( 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;
        }
    }
}

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.

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. Open up a display list in (GPU) memory
4. The coordinates, etc. end up in memory instead of being sent to the display
5. Stop storing the numbers in the DL
6. Pull all the coordinates, etc. from memory, just as if the code to generate them had been executed here

Calling up the Display List in Display( )

Creating the Display List in InitLists( )

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

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.