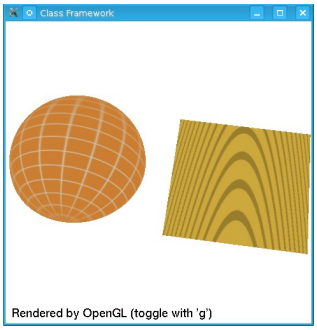
CS241 Project 4

Synopsis

Enhance the triangle scan conversion algorithm with anti-aliased, perspective-correct texture maps.

Instructions



This project adds texture mapping to the previous projects. (That is, the clipper and triangle scan-conversion projects.) A new **scene.cpp** is provided which defines a simple scene with objects and their associated texture maps. You are to implement perspective-correct texture mapping and anti-aliasing (both magnification via bilinear interpolation, and minification via MIPmaps.

Here is an outline of the steps involved:

1. **Texture retrieval:** Each object in the scene data structure has an attribute named **texture** of type **Texture***. When it is non-NULL, it contains a texture. Calculate and store the MIPmap for anti-aliasing purposes.
2. **Vertex transformations:** When a vertex is read from the scene data structure, and transformed, the texture coordinates (attributes u and v) are now read at the same time and associated with the transformed vertices.
3. **In the clipper:** The input and output vertices of the clipper now have associated texture coordinates. Whenever a vertex is copied to output, the associated texture attribute is also copied with it. When a new output vertex is created (via interpolation at an edge's intersection of a window boundary), the new vertex gets a new texture coordinate calculated via the same interpolation applied to the edge's texture coordinates.
4. **Homogeneous division:** When a transformed point (x,y,z,w) with associated texture coordinate (u,v) as about to undergo the homogeneous division, the texture coordinate must also be modified for perspective-corrected interpolation:
   $(u,v) \rightarrow (u/w, v/w, 1/w)$, and $(x,y,z,w) \rightarrow (x/w, y/w, z/w)$.
5. **Scan-conversion:** When the vertices are reordered, so too are their texture coordinates. As the x and z values is interpolated up each edge, so too are the texture coordinates. And finally, as the pixel's depths are interpolated across a scan-line, so too are the texture coordinates.
6. **Lookup:** Finally, when a pixel is calculated and its texture attribute is known, the texture coordinate is used to lookup the color to be placed at that pixel. First, let the interpolated texture coordinate be called (u',v',w'), and calculate $(u,v) = (u'/w', v'/w')$. In order to implement a repeating texture, replace each of u and v with their fractional portion. Then:
   a) **First pass (no-anti-aliasing):**
   Calculate the single texel index
   (i,j) = (floor(u*tex->width), floor(v*tex->height))`
   and get a pointer to the color (as an array of three floats) by calling
   rgb = tex.texel(i,j,0)
   and set the color of the pixel via OpenGL calls
   glColor3fv(rgb) // Set color
   glVertex2f(x,y) // Draw pixel
   b) **Second pass (with anti-aliasing):** Calculate an estimate of the pixel's quad size in texture coordinates, and use this to implement anti-aliasing. Use bilinear interpolation of four texture look-ups for any (u,v) lookups in a MIPmap level, and use a linear map between two adjacent levels when appropriate.
   c) **Third pass (optional suggestion):** After you get texturing working correctly, go back to step 4 and disable the perspective correction and notice the effect. (Then re-enable it please.)

Project Report

Submit a **succinct** report. Place it in a text file named **report.txt** or **report.doc** or **report.odt** if you (unnecessarily) insist on producing a formated document.

To submit

Use Moodle to submit your project via the “CS241 Project 4” link on the class web page. Be sure to submit a **zip** of any source files that you have changed since starting this sequence of projects. If you have added any files of you own, be sure to also include the *.vcproj file.

Grading Basis

Grading will be on a 100 point basis, with the following distribution:

- **Texture:** 30%, for the proper implementation of basic texturing.
- **Anti-alaising:** 20%, for MIPmap or SAT for minification anti-aliasing.
- **Anti-aliasing:** 20% for bilinear interpolation for magnification anti-aliasing.
- **Code:** 20%, judgment based on looking at the code for reasonable coding and commenting practices.
- **Project report:** 10%, judgment based on how succinctly and accurately the project report describes your implementation.