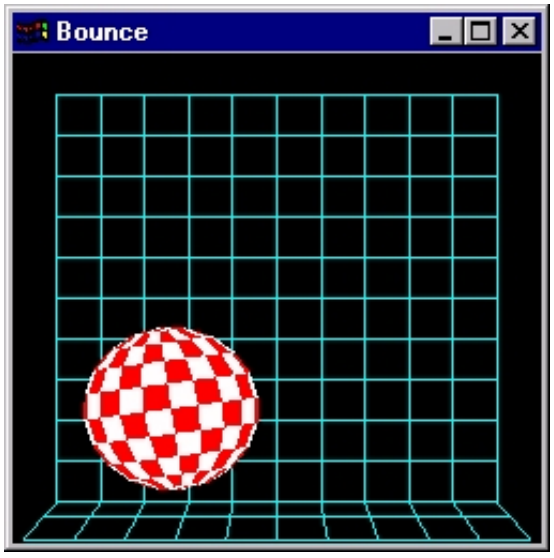
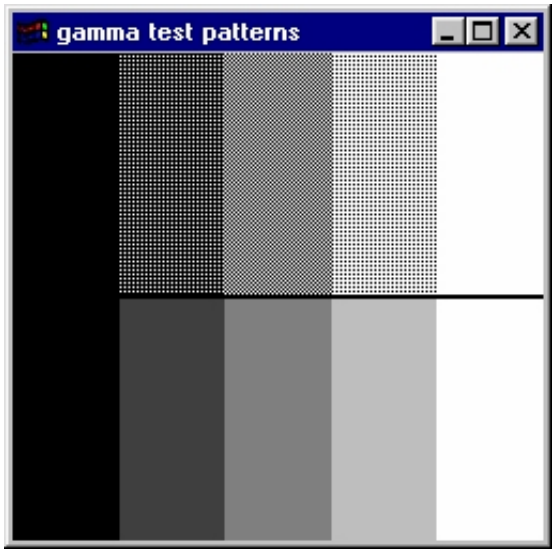
mesademos.zip



A simple bouncing ball demo (color index mode only).

Source code: bounce.c.

Snapshots: color index (shown).



Draw test patterns to help determine correct gamma value for a display. When the intensities in the top row nearly match the intensities in the bottom row you've found the right gamma value.

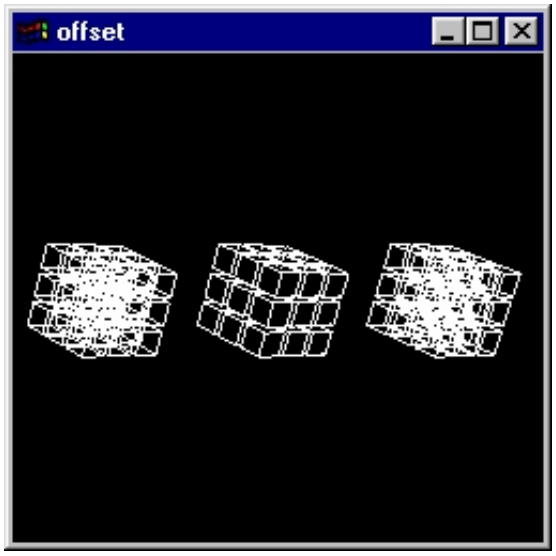
Source code: gamma.c.

Snapshots: gamma (shown).

Simple program with rotating 3-D gear wheels.

Source code: gears.c.

Snapshots: gears (shown).

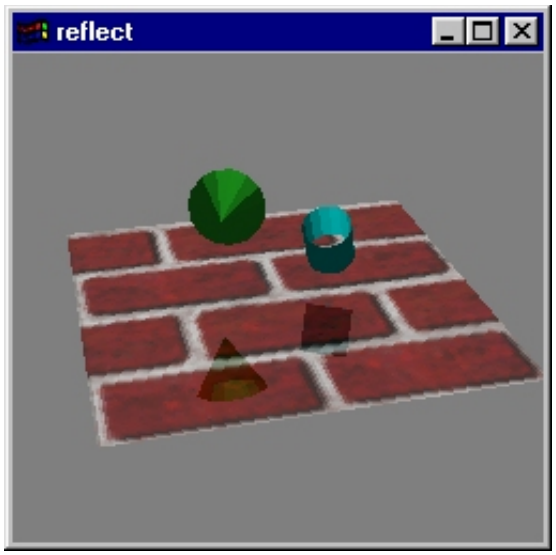


Uses PolygonOffset to draw hidden-line images. PolygonOffset shifts the z values of polygons an amount that is proportional to their slope in screen z. This keeps the lines, which are drawn without displacement, from interacting with their respective polygons, and thus eliminates line dropouts. The left image shows an ordinary antialiased wireframe image. The center image shows an antialiased hidden-line image without PolygonOffset. The right image shows an antialiased hidden-line image using PolygonOffset to reduce artifacts.

Source code: offset.c.

Snapshots: offset (shown).

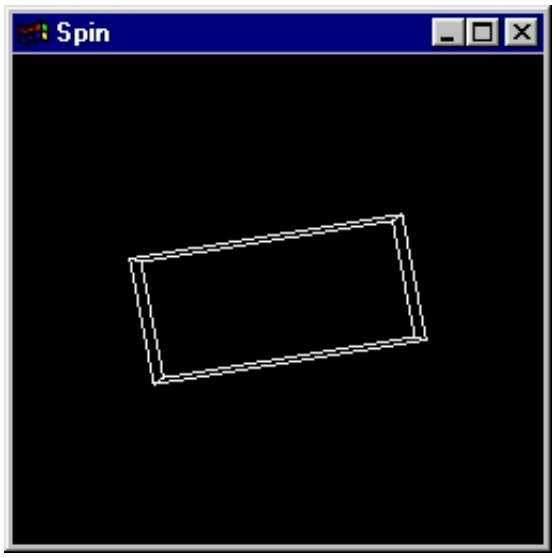
Reflection demo. The basic steps are: 1. Render the reflective object (a polygon) from the normal viewpoint, setting the stencil planes = 1. 2. Render the scene from a special viewpoint: the viewpoint which is on the opposite side of the reflective plane. Only draw where stencil = 1. This draws the objects in the



reflective surface. 3. Render the scene from the original viewpoint. This draws the objects in the normal fashion. Use blending when drawing the reflective, textured surface.

Source code: reflect.c.

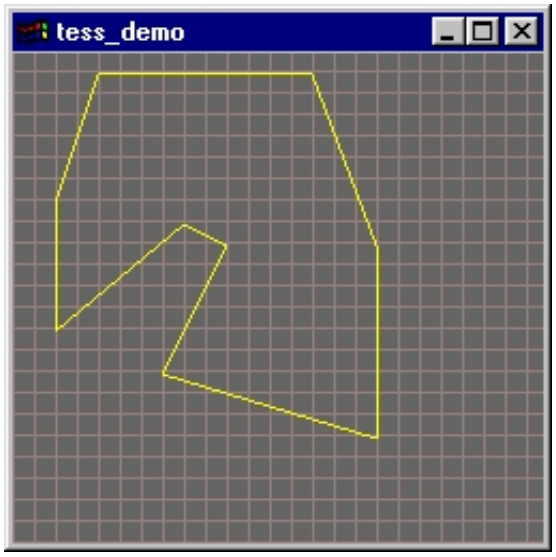
Snapshots: reflect (shown).



A very simple wireframe spinning box.

Source code: spin.c.

Snapshots: spin (shown).

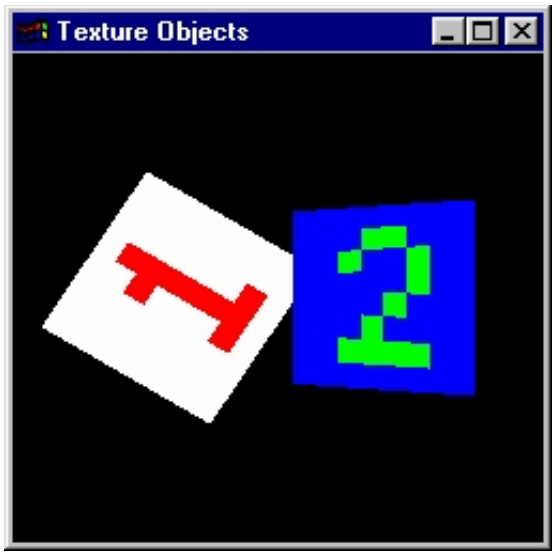


Dynamic tesselation demonstration program.

Source code: tess_demo.c.

Snapshots: tess_demo (shown).

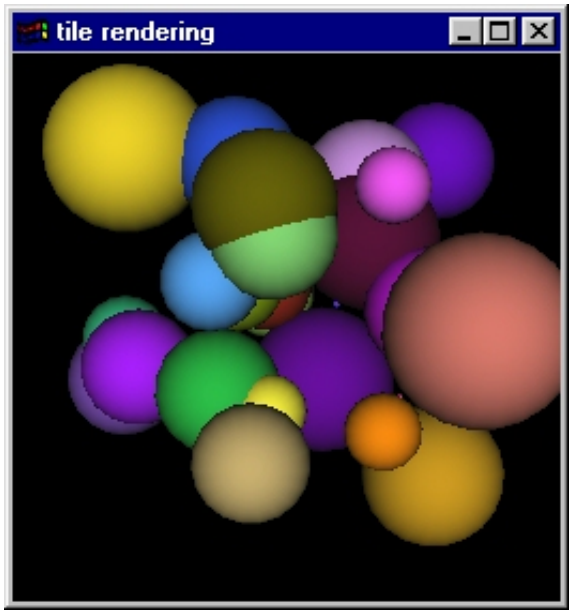
Example of using the 1.1 texture object functions. Also, this demo



utilizes Mesa's fast texture map path.

Source code: texobj.c.

Snapshots: texobj (shown).



Test/demonstration of tile rendering utility library. This library allows one to render arbitrarily large images with OpenGL. The basic idea is to break the image into tiles which are rendered one at a time. The tiles are assembled together to form the final, large image. Tiles can be of any size.

Source code: trdemo.c.

Snapshots: trdemo (shown).