A guide to the shader examples

This guide describes programs very similar to those described in the "Shaders" chapter of *Computer Graphics: Principles and Practice*. There have been a few changes to G3D (which we use to load and execute the shaders) and to OpenGL since we wrote the book; we've also slightly generalized the examples by adding choosable colors and a few other features. They are, however, substantially the same. All the shader demos share a single user interface that lets the user select a diffuse and specular color, adjust shininess, etc. Naturally, for the Gouraud shading demo, only the diffuse options have any effect, and for the XToon shader demo, none of the interface elements get used at all. But this simplifies the explanation somewhat.

Large-scale changes

- OpenGL changed and made names like `gl_Vertex` and `gl_Normal` no longer predefined, and in fact no longer usable; G3D was altered to define `g3d_Vertex` and `g3d_Normal` instead, serving exactly the same role. The output variable for the vertex shader, however, is still `gl_Position`, although this too may change in the future.
- There's now a drop-down menu for selecting the diffuse color and specular color in the examples
- A few variable names have changed slightly
- We've loaded a teapot rather than an icosahedron as the standard model.

Large scale program structure

There's a single C++ file containing code for a single class, App, which does all the heavy lifting.

App: The application that loads up a model, loads the vertex- and fragment-shaders to be used, establishes the lighting and camera position for the scene to be shown (which is always a single teapot), displays a simple color-choice and value-selection user interface, and communicates user-settings from the UI to the shaders so that they can apply those to the rendering of the teapot. The application also displays a click-and-draggable widget in the form of three rings and three arrows along the three coordinate axes. Dragging on an arrow moves the object in that directions; dragging on the circle in the xy-plane rotates the object in the xy-plane, and similarly for the other two circles.

Commentary on the code

What follows is the code for the App class, and then shader code for the Gouraud shading program, detailing differences from the book's version.

The App Class (for the shader projects)

```
/**

  Example of using G3D shaders and GUIs.

 */
#include <G3D/G3DAll.h>
#include <GLG3D/GLG3D.h>

class App : public GApp {
private:
    shared_ptr<Lighting>            lighting; // a representation of the lights in a scene
    shared_ptr<ArticulatedModel>    model; // G3D utility class for representing polygonal models

    shared_ptr<Shader>              gouraudShader; // A class representing a piece of shader code and its compilation

    float                           diffuseScalar; // The k_d term in the Phong lighting model, or the reflectance
                                                   // of the diffuse surface for the Gouraud/Lambertian model
    int                             diffuseColorIndex; // Which of the several offered colors to use

    float                           specularScalar; // The k_s term in the Phong model – unused in this shader/program
    int                             specularColorIndex; // Also unused

    float                           reflect; // Also unused
    float                           shine; // Also unused

    ////////////////////////////////////
    // GUI

    /** For dragging the model */
    shared_ptr<ThirdPersonManipulator>   manipulator; // A 3-ring, 3-arrow controller
    Array<GuiText>                  colorList; // Names for the colors to be offered to the user in the UI

    void makeGui();
    void makeColorList();
    void makeLighting();
    void configureShaderArgs(const Lighting::Ref localLighting, Args& args);

public:
   // Initialize with a 60% reflective surface; the specular, reflect, and shine numbers are unused,
   // but they get used in the Phong shader, which uses the same “main.cpp” program.
```

```
    App() : diffuseScalar(0.6f), specularScalar(0.5f), reflect(0.1f), shine(20.0f) {}

    virtual void onInit();
    virtual void onGraphics3D(RenderDevice* rd, Array<shared_ptr<Surface> >& surface3D);
};


// Initialization. The first large block loads up the model, positioning it, scaling it to a good size,
// and removing any material properties that happen to be in the model-file.
//
// The second block sets up lighting, a list of colors used in the GUI, and then the GUI itelsf.
// The third section picks a particular starting color, puts the camera in a good spot, and adds the
// 3-rings-3-arrows manipulator that lets us move the teapot around.
void App::onInit() {
    createDeveloperHUD(); // A small set of controls that we initially suppress, but the user can show
                          // by pressing F11. When displayed, this allows you to take snapshots of the display
                          // record movies, display the number of frames shown per second, reload shader code,
                          // and several other things.
    window()->setCaption("Pixel Shader Demo");

    gouraudShader = Shader::fromFiles("gouraud.vrt", "gouraud.pix"); // Read the shader cord and compile it.
    // Load up the teapot.
    ArticulatedModel::Specification spec;
    spec.filename = System::findDataFile("teapot/teapot.obj");
    spec.scale = 0.015f;
    spec.stripMaterials = true;
    spec.preprocess.append(ArticulatedModel::Instruction(Any::parse("setCFrame(root(), Point3(0, -0.5, 0));")));
    model = ArticulatedModel::create(spec);

    makeLighting();
    makeColorList();
    makeGui();

    // Color 1 is red
    diffuseColorIndex = 1;
    // Last color is white
    specularColorIndex = colorList.size() - 1;
    // Set the camera to look at the teapot
    m_debugCamera->setPosition(Vector3(1.0f, 1.0f, 2.5f));
    m_debugCamera->setFieldOfView(45 * units::degrees(), FOVDirection::VERTICAL);
    m_debugCamera->lookAt(Point3::zero());
```

```
    // Add axes for dragging and turning the model
    manipulator = ThirdPersonManipulator::create();
    addWidget(manipulator);

    // Turn off the default first-person camera controller and developer UI
    m_debugController->setEnabled(false);
    developerWindow->setVisible(false); // F11 turns it back on
    developerWindow->cameraControlWindow->setVisible(false); // F2 turns it back on
    showRenderingStats = false;
}


void App::onGraphics3D(RenderDevice* rd, Array<shared_ptr<Surface> >& surface3D) {
    // Bind the main frameBuffer
    rd->pushState(m_frameBuffer); {
        rd->clear(); // Clear out the framebuffer

        rd->setProjectionAndCameraMatrix(m_debugCamera->projection(), m_debugCamera->frame());

        // Add in an environment map, which is used only in the phong lighting example.
        Draw::skyBox(rd, lighting->environmentMapTexture, lighting->environmentMapConstant);

        rd->pushState(); {
            Array< shared_ptr<Surface> > mySurfaces;
            // Pose our model based on the manipulator axes
            model->pose(mySurfaces, manipulator->frame());

            // Set up shared arguments
            Args args;
            configureShaderArgs(lighting, args);

            // Send model geometry to the graphics card
            CFrame cframe;
            for (int i = 0; i < mySurfaces.size(); ++i) {

                // Downcast to UniversalSurface to access its fields
                shared_ptr<UniversalSurface> surface = dynamic_pointer_cast<UniversalSurface>(mySurfaces[i]);
                if (notNull(surface)) {
                    surface->getCoordinateFrame(cframe);
                    rd->setObjectToWorldMatrix(cframe);
                    surface->gpuGeom()->setShaderArgs(args);
```

```
                    // (If you want to manually set the material properties and vertex attributes
                    // for shader args, they can be accessed from the fields of the gpuGeom.)

                    rd->apply(gouraudShader, args); // Finally! We tell it to use our shader, with the
                                                    // arguments we've previously set.
                }
            }
        } rd->popState();

        // Render other objects, e.g., the 3D widgets
        Surface::Environment env;
        Surface::render(rd, m_debugCamera->frame(), m_debugCamera->projection(), surface3D, surface3D, lighting, env);
    } rd->popState();

    // Perform gamma correction, bloom, and SSAA, and write to the native window frame buffer
    m_film->exposeAndRender(rd, m_debugCamera->filmSettings(), m_colorBuffer0, 1);

}


// Put values in an Arg data structure for transfer to the GPU (about 10 lines above here)
void App::configureShaderArgs(const shared_ptr<Lighting> lighting, Args& args) {
    const shared_ptr<Light>&  light         = lighting->lightArray[0];
    const Color3&       diffuseColor  = colorList[diffuseColorIndex].element(0).color(Color3::white()).rgb();
    const Color3&       specularColor = colorList[specularColorIndex].element(0).color(Color3::white()).rgb();


    // Viewer
    args.setUniform("wsEyePosition",       m_debugCamera->frame().translation);

    // Lighting
    args.setUniform("wsLight",             light->position().xyz().direction());
    args.setUniform("lightColor",          light->color);
    args.setUniform("ambient",             Color3(0.3f)); // Ambient light unused in our examples.
    args.setUniform("environmentMap",      lighting->environmentMapTexture); // Environment map used in Phong shading
    args.setUniform("environmentConstant", lighting->environmentMapConstant);

    // UniversalMaterial
    args.setUniform("diffuseColor",        diffuseColor);
    args.setUniform("diffuse",             diffuseScalar);
```

```
    args.setUniform("specularColor",        specularColor);
    args.setUniform("specularScalar",       specularScalar);

    args.setUniform("shine",                shine);
    args.setUniform("reflect",              reflect);
}


// OK, this is kind of a hack. To make bars of constant color for the user to select,
// we actually display some text in a font where the letter "g" is a filled rectangle of color. So "gggg" is a
// bar of color.
void App::makeColorList() {
    GFont::Ref iconFont = GFont::fromFile(System::findDataFile("icon.fnt"));

    // Characters in icon font that make a solid block of color
    static const char* block = "gggggg";

    float size = 18;
    int N = 10;
    colorList.append(GuiText(block, iconFont, size, Color3::black(), Color4::clear()));
    for (int i = 0; i < N; ++i) {
        colorList.append(GuiText(block, iconFont, size, Color3::rainbowColorMap((float)i / N), Color4::clear()));
    }
    colorList.append(GuiText(block, iconFont, size, Color3::white(), Color4::clear()));
}

// Build a GUI that offers the user sliders to select how much diffuse color (and what diffuse color) to use,
// and similarly for the specular color (unused in the Gouraud demo), and the mirror reflectance and
// the smoothness of the Phong peak (also unused in the Gouraud demo).
void App::makeGui() {
    GuiWindow::Ref gui = GuiWindow::create("UniversalMaterial Parameters");
    GuiPane* pane = gui->pane();

    pane->beginRow();
    pane->addSlider("Lambertian", &diffuseScalar, 0.0f, 1.0f);
    pane->addDropDownList("", colorList, &diffuseColorIndex)->setWidth(80);
    pane->endRow();

    pane->beginRow();
    pane->addSlider("Glossy",    &specularScalar, 0.0f, 1.0f);
    pane->addDropDownList("", colorList, &specularColorIndex)->setWidth(80);
    pane->endRow();
```

```
    pane->addSlider("Mirror",     &reflect, 0.0f, 1.0f);
    pane->addSlider("Smoothness", &shine, 1.0f, 100.0f);

    gui->pack();
    addWidget(gui);
    gui->moveTo(Point2(10, 10));
}

// We build a lighting-specification (one light source plus an environment map showing a cloudy sky
// with a mostly-obscured sun), from which we can (in the last line) create a “Lighting” object, which
// is what we can pass to the GPU.

void App::makeLighting() {
    Lighting::Specification spec;
    spec.lightArray.append(Light::directional("Light", Vector3(1.0f, 1.0f, 1.0f), Color3(1.0f), false));

    // The environmentMap is a cube of six images that represents the incoming light to the scene from
    // the surrounding environment.  G3D specifies all six faces at once using a wildcard and loads
    // them into an OpenGL cube map.
    spec.environmentMapConstant = 1.0f;
    spec.environmentMapTexture.filename   = FilePath::concat(System::findDataFile("nooncloudes"), "nooncloudes_*.png");
    spec.environmentMapTexture.dimension  = Texture::DIM_CUBE_MAP;
    spec.environmentMapTexture.settings   = Texture::Settings::cubeMap();
    spec.environmentMapTexture.preprocess = Texture::Preprocess::gamma(2.1f);
    // Reduce memory size required to work on older GPUs
    spec.environmentMapTexture.preprocess.scaleFactor = 0.25f;
    spec.environmentMapTexture.settings.interpolateMode = Texture::BILINEAR_NO_MIPMAP;

    lighting = Lighting::create(spec);
}

G3D_START_AT_MAIN();

int main(int argc, char** argv) {
// This seeks to find the shader code in G3D’s data directory if you run the program wrong.
// It’ll probably find it, but then fail because of name-mismatches.
#   ifdef G3D_WINDOWS
      if (! FileSystem::exists("gouraud.pix", false) && FileSystem::exists("G3D.sln", false)) {
          // The program was started from within Visual Studio and is
          // running with cwd = G3D/VC10/.  Change to
          // the appropriate sample directory.
```

```
            chdir("../samples/pixelShader/data-files");
        } else if (FileSystem::exists("data-files")) {
            chdir("data-files");
        }
#   endif

    return App().run();
}
```

The vertex shader (Gouraud project)

```
#version 120 // -*- c++ -*-
/**
  Per-pixel Gouraud Shading.

 */
/* G3D now always defines these, so we have to do so as well */
attribute vec4 g3d_Vertex; // Note that the “gl” prefix is mostly replaced by “g3d”.
attribute vec3 g3d_Normal;

/** How well-lit is this vertex? */
varying float gouraudFactor;

/** Unit world space direction to the (infinite, directional) light source */
uniform vec3 wsLight;

/** Non-unit surface normal in world space */
varying vec3 wsInterpolatedNormal; // now declared outside of "main"; // Formerly called wsNormal

void main(void) {
    wsInterpolatedNormal = normalize(g3d_ObjectToWorldNormalMatrix * g3d_Normal);
    gouraudFactor = dot(wsInterpolatedNormal, wsLight);

    gl_Position = g3d_ModelViewProjectionMatrix * g3d_Vertex; // gl_Position remains as a gl_-prefixed name
}
```

The fragment shader (Gouraud project)

```
#version 120 // -*- c++ -*-
/**
 Per-pixel Gouraud Shading.

 */

/** Diffuse/ambient surface color */
uniform vec3 diffuseColor;

/** Intensity of the diffuse term. */
uniform float diffuse;
```

```
/** Color of the light source */
uniform vec3 lightColor;

/** dot product of surf normal with light */
varying float gouraudFactor;

void main() {
    gl_FragColor.rgb =
        diffuse  * diffuseColor  * ((max(gouraudFactor, 0.0) * lightColor));
}
```