G4G_Basic Master Branch:

```
//renderer object that does the rendering with model, view, and projection matrix setup
//render() method
renderer.h

//Defined in basics.cpp
//Creates a simple image
int myTexture()

//Child class of renderer
Class QuadRenderer:
     public QuadRenderer()
     protected indices[]

void framebuffer_size_callback()

void drawIMGUI()

//Error checking and minimal
int main()
     //Has a vector of render objects
```

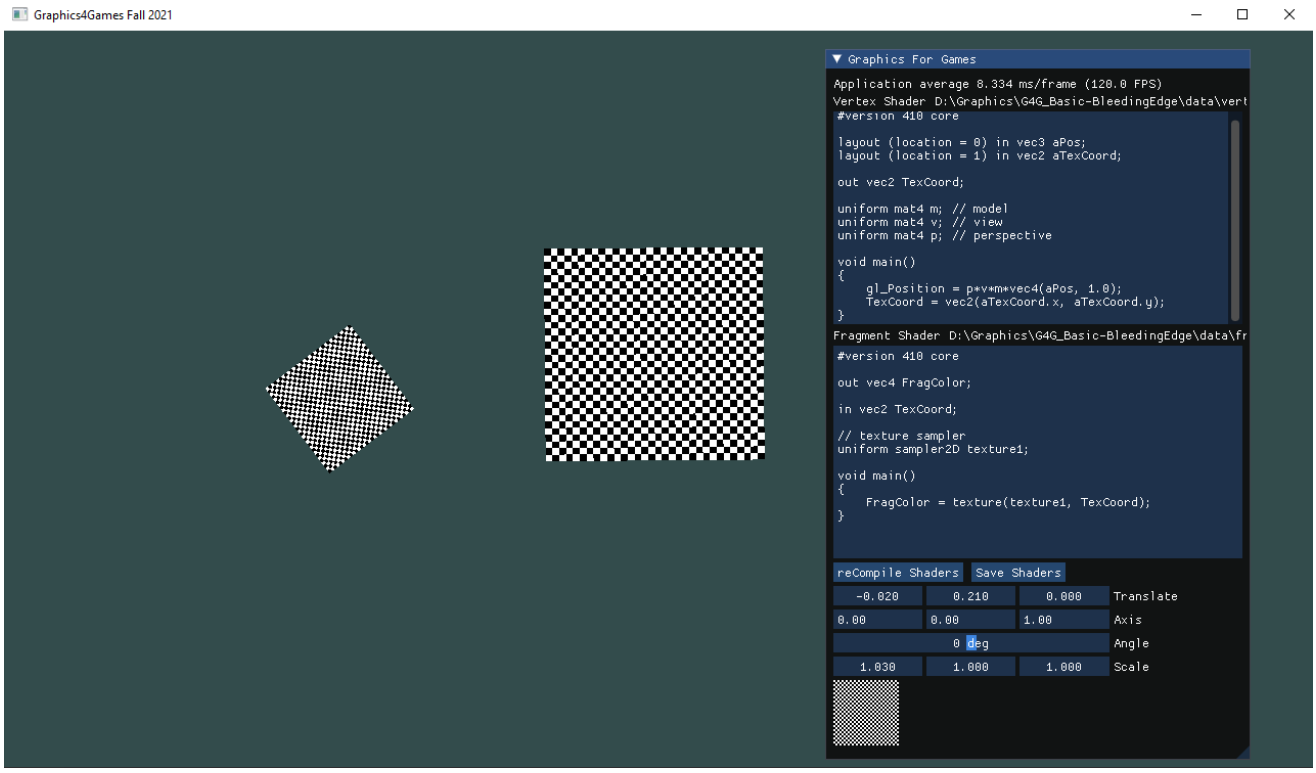
G4G_Basic BleedingEdge Branch:

- **Build**

```
#ifdef sandbox
int myTexture()
//RayTracing algorithm from RayTracing.cpp
//Resources: https://www.youtube.com/watch?v=gBPNO6ruevk
int RayTracer()

void setupTextures()

//Draws an IMGUI window with one ray-traced image
void drawIMGUI()

int main()

void framebuffer_size_callback()

#else
//Draws a single quad
```

- **Build2**

//Similar to the version inside the master branch with some easter eggs uncommented
//No texture showing: have to change the vs/fs



//Save shader not working

- **Build3**
- **Main.cpp**

```
//Change Chapter# to Chapter0(Base class), Chapter1 etc.
Chapter# myDemo;
int main() {

	//Initialize myDemo
	myDemo.start();

	glfwSetFramebufferSizeCallback(window, callback_function);
	while(!glfwWindowShouldClose(window)) {

		//Update
		myDemo.update()
	}
}
```

- **SceneGraph.h / SceneGraph.cpp**

```
//Representation of a whole scene

//Go through the tree structure and render each of the children
void treeNode::traverse(glm::mat4 treeModelMat, glm::mat4
viewProjection, double deltaTime, SceneGraph *sg)

//Erase method for both treeNode and SceneGraph
void treeNode::purgeRenderer(Renderer *x)
void SceneGraph::purgeRenderer(Renderer *x)

//A SceneGraph object contains the following
//Camera, light, a tree of renderers, regular/shadow render pass
//   |                          |
//   |                          |
//Persp/Ortho                   |
//                              |
                                |
```

- **renderer.h / renderer.cpp** <-|

```
//A renderer object takes care of the actual drawing of
//TorusModel/CubeModel/TriangleModel/SkyboxModel/iCubeModel/SphereModel
//One can set model matrix, rotate, translate, and scale the model

//The inherited "tree" transform is in the view and
//combined the projection and view into a  vpMat
void Renderer::render(glm::mat4 treeMat, glm::mat4 vpMat...)

void Renderer::setupColorAttrib()
```

- **Chapter0.cpp**

```
//Create a window
```

- **Chapter1.cpp**

```
//Draws a single triangle
//From triangleModel.cpp
static void drawMyGUI(SceneGraph* sg,Renderer *myRenderer)

void Chapter1::start()

void Chapter1::update(double deltaTime) {
      // draw the triangle directly, no camera, no perspective
      triangle->render(...);

      drawMyGUI(&scene, 0L);
}

// housekeeping, remove all the shaders, materials and renderers created
void Chapter1::end()

void Chapter1::callback(GLFWwindow* window, int width, int height)
```

- **Chapter1a.cpp**

```
//Draws a cube with a floor and skybox

...

//Create each side of the cube and add the faces to the scene
static void quadCube()

void Chapter1::start() {
      //skybox is special and doesn't belong to the SceneGraph
      mySky = new SkyboxModel(...);

      quadCube();
}

 ...
```

- **Chapter2.cpp**

```
//Draws a cube with a floor and skybox

...

//Create each side of the cube and add the faces to the scene
static void quadCube()

void animateNodes(treeNode** nodes, double time)

void Chapter1::start() {
     //skybox is special and doesn't belong to the SceneGraph
     mySky = new SkyboxModel(...);

     quadCube();
}

void Chapter1::update() {
     animateNodes();
}
 ...
```

- **General Questions:**
  - mvp = vpMat * treeMat * modelMatrix?
  - Build the solutions through Visual Studio first?