CITS3003 Graphics and Animation
Project Report (Part 1)

Ridge Shrubsall (21112211)

Functionality

All of the main functionality works and has been thoroughly tested. My partner Even and I worked on tasks A-I together, and I implemented object duplication for task J and loading/saving scenes for task K as my individual functionality.

Implementation

a) Camera rotation

```
353     mat4 rot = RotateX(camRotUpAndOverDeg) * RotateY(camRotSidewaysDeg);
354     view = Translate(0.0, 0.0, -viewDist) * rot;
```

The display function needed to be modified in order to rotate the camera. This was done by adding the appropriate X and Y rotations using the camera angle variables.

b) Object rotation

```
334     mat4 rot = RotateX(sceneObj.angles[0]) * RotateY(sceneObj.angles[1]) *
RotateZ(sceneObj.angles[2]);
335     mat4 model = Translate(sceneObj.loc) * rot * Scale(sceneObj.scale);
```

Similarly, the drawMesh function needed to be modified in order to rotate the objects. This was done by adding the appropriate X, Y and Z rotations using the angles array for each object.

```
537     } else if (id == 55 && currObject >= 0) {
538         setToolCallbacks(adjustAngleYX, mat2(400.0, 0.0, 0.0, 400.0),
539                 adjustAngleZTexScale, mat2(-400.0, 0.0, 0.0, 15.0));
```

The rotation tool's directions also needed to be changed in order to match the sample solution. This required the X and Z rotations to be inverted.

c) Lighting and shininess adjusters

```
466 static void adjustAmbientDiffuse(vec2 ad) {
467     SceneObject *obj = &sceneObjs[toolObj];
468     obj->ambient = max(0.0f, obj->ambient + ad[0]);
469     obj->diffuse = max(0.0f, obj->diffuse + ad[1]);
470 }
471
472 static void adjustSpecularShine(vec2 ss) {
473     SceneObject *obj = &sceneObjs[toolObj];
474     obj->specular = max(0.0f, obj->specular + ss[0]);
475     obj->shine = max(0.0f, obj->shine + ss[1]);
476 }
```

Adjustment functions for the amount of ambient, diffuse and specular light as well as the amount of shine needed to be added.

```
485         } else if (id == 20) {
487             toolObj = currObject;
488             setToolCallbacks(adjustAmbientDiffuse, mat2(1.0, 0.0, 0.0, 1.0),
489                     adjustSpecularShine, mat2(1.0, 0.0, 0.0, 10.0));
```

A new menu item for these adjusters was also added to the materialMenu function.

e) Horizontal reshaping

```
664     GLfloat left, right, bottom, top;
665
669     if (width < height) {
670         left = -nearDist;
671         right = nearDist;
672         bottom = -nearDist * (float) height / (float) width;
673         top = nearDist * (float) height / (float) width;
674     } else {
675         left = -nearDist * (float) width / (float) height;
676         right = nearDist * (float) width / (float) height;
677         bottom = -nearDist;
678         top = nearDist;
679     }
```

The reshape function needed to be modified to scale the viewport in the same way as when the window is resized vertically. This was done by using the height-to-width ratio to scale the bottom and top planes when the width is less than the height.

d) Close-up view

```
663     GLfloat nearDist = 0.04;
(...)
681     projection = Frustum(left, right, bottom, top, 0.2, 500.0);
```

To provide more "close-up" views of objects, the near distance was also scaled by a factor of 5 to increase the camera's field of view.

```
27 static float viewDist = 7.5;
(...)
215 static void doRotate() {
216     setToolCallbacks(adjustCamSideViewDist, mat2(400.0, 0.0, 0.0, -10.0),
217             adjustCamSideUp, mat2(400.0, 0.0, 0.0, -90.0));
218 }
```

The initial view distance and Z movement speed were also scaled to compensate.

g) Per-fragment lighting

In order to do per-fragment lighting, the lighting calculations were moved from the vertex shader into the fragment shader using the code from Lecture 17 as a guide. No changes to the C++ source code needed to be made.

i) Second light

```
299     addObject(55);  // Sphere for the second light
300     SceneObject *lightObj2 = &sceneObjs[2];
301     lightObj2->loc = vec4(-2.0, 1.0, -1.0, 1.0);
302     lightObj2->scale = 0.1;
303     lightObj2->texId = 0;  // Plain texture
304     lightObj2->brightness = 0.5;
```

To add a second light to the scene, a new sphere object is created and positioned opposite to the first one.

```
356     SceneObject lightObj1 = sceneObjs[1];
357     vec4 lightPosition1 = view * lightObj1.loc;
358     SceneObject lightObj2 = sceneObjs[2];
359     vec4 lightPosition2 = rot * lightObj2.loc;
```

The second light should be directional ($w = 0$) and thus only the camera rotation needs to be applied.

```
361     glUniform4fv(glGetUniformLocation(shaderProgram, "LightPosition1"), 1,
lightPosition1); CheckError();
362     glUniform4fv(glGetUniformLocation(shaderProgram, "LightPosition2"), 1,
lightPosition2); CheckError();
```

The positions of the two lights are then passed into the vertex shader. The lighting calculations are then duplicated in order to handle the second light.

f) Light reduction

h) Specular highlights

```
363     glUniform3fv(glGetUniformLocation(shaderProgram, "LightColor1"), 1,
lightObj1.rgb); CheckError();
364     glUniform3fv(glGetUniformLocation(shaderProgram, "LightColor2"), 1,
lightObj2.rgb); CheckError();
365     glUniform1f(glGetUniformLocation(shaderProgram, "LightBrightness1"),
lightObj1.brightness); CheckError();
366     glUniform1f(glGetUniformLocation(shaderProgram, "LightBrightness2"),
lightObj2.brightness); CheckError();
(…)
371         vec3 rgb = obj.rgb * obj.brightness * 2.0;
```

The colour and brightness of the two lights are now passed into the fragment shader separately rather than being multiplied into *rgb*.

```
38     vec3 specular1 = Ks1 * LightBrightness1 * SpecularProduct;
39     vec3 specular2 = Ks2 * LightBrightness2 * SpecularProduct;
```

The specular component should be independent of the light's colour, thus it is only multiplied by the brightness.

```
52     float len = 0.01 + length(fL1);
53     vec4 color = vec4(globalAmbient + ((ambient1 + diffuse1) / len) + ambient2 +
diffuse2, 1.0);
54     fColor = color * texture2D(texture, texCoord * texScale) + vec4((specular1 /
len) + specular2, 1.0);
```

The specular component should also be independent of the texture's colour, thus it is added on at the very end.

The intensity of the first light is also scaled down by its distance.

Individual implementation

j) Object duplication

```
509 static void duplicateObject(int id) {
510     if (nObjects == maxObjects) return;
511
512     sceneObjs[nObjects] = sceneObjs[id];
513     toolObj = currObject = nObjects++;
514     setToolCallbacks(adjustLocXZ, camRotZ(),
515             adjustScaleY, mat2(0.05, 0.0, 0.0, 10.0));
516     glutPostRedisplay();
517 }
(…)
540     } else if (id == 90 && currObject >= 0) {
541         duplicateObject(currObject);
(…)
633     glutAddMenuEntry("Duplicate object", 90);
```

To duplicate an object, the last object just has to be copied into the next position of the scene objects array. The number of objects is then incremented, and the rotation tool is then selected on the new object.

k) Scene loading/saving

```
68 const int numSaves = 30;
69 const int saveHeader = ('S' | 'A' << 8 | 'V' << 16 | 'E' << 24);
(...)
549 static void loadMenu(int id) {
550     char fileName[256];
551     sprintf(fileName, "slot%d.sav", id);
552
553     FILE *file = fopen(fileName, "rb");
554     if (file == NULL) {
555         fprintf(stderr, "Error: Could not open '%s' for reading\n", fileName);
556         return;
557     }
558
559     int header;
560     fread(&header, sizeof(int), 1, file);
561     if (header != saveHeader) {
562         fprintf(stderr, "Error: Invalid save file header");
563         fclose(file);
564         return;
565     }
566     fread(&viewDist, sizeof(float), 1, file);
567     fread(&camRotSidewaysDeg, sizeof(float), 1, file);
568     fread(&camRotUpAndOverDeg, sizeof(float), 1, file);
569     fread(&nObjects, sizeof(int), 1, file);
570     memset(sceneObjs, 0, sizeof(SceneObject) * nObjects);
571     fread(sceneObjs, sizeof(SceneObject), nObjects, file);
572
573     currObject = nObjects - 1;
574     toolObj = -1;
575     doRotate();
576
577     fclose(file);
578 }
579
580 static void saveMenu(int id) {
581     char fileName[256];
582     sprintf(fileName, "slot%d.sav", id);
583
584     FILE *file = fopen(fileName, "wb");
585     if (file == NULL) {
586         fprintf(stderr, "Error: Could not open '%s' for writing\n", fileName);
587         return;
588     }
589
590     fwrite(&saveHeader, sizeof(int), 1, file);
591     fwrite(&viewDist, sizeof(float), 1, file);
592     fwrite(&camRotSidewaysDeg, sizeof(float), 1, file);
593     fwrite(&camRotUpAndOverDeg, sizeof(float), 1, file);
594     fwrite(&nObjects, sizeof(int), 1, file);
595     fwrite(sceneObjs, sizeof(SceneObject), nObjects, file);
596
597     fflush(file);
598     fclose(file);
599 }
(...)
617     char saveMenuEntries[numSaves][128];
618     for (int i = 0; i < numSaves; i++) {
619         sprintf(saveMenuEntries[i], "Slot %d", i+1);
620     }
621     int loadMenuId = createArrayMenu(numSaves, saveMenuEntries, loadMenu);
622     int saveMenuId = createArrayMenu(numSaves, saveMenuEntries, saveMenu);
(...)
635     glutAddSubMenu("Load scene", loadMenuId);
636     glutAddSubMenu("Save scene", saveMenuId);
```

The load/save functions store scenes in a binary format, which includes the position of the camera and the state of each object. A header is written at the start of the file which serves as a safety check.

[Last minute bug fix: the fopen call needs the binary mode flag (‘b’) in order to work under Windows!]

Reflection

I felt that the tasks in the project had a rather steep learning curve, and so it took longer than I thought it would to complete. In some parts, I had to resort to trial and error in order to figure out what was going on but I managed to implement all of the tasks in the end.