



UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SCHOOL OF COMPUTING
Faculty of Engineering

SCSV4543-02

ADVANCED COMPUTER GRAPHICS

(Session 2020/2021 Semester 2)



School of Computing

Universiti Teknologi Malaysia

ASSIGNMENT 2

VIEWING, SHADING, LIGHTING AND TEXTURE

PREPARED BY:

GROUP MEMBERS	 WAN NUR KHALISHAH BINTI MASRY	 SITI NURAZALINA BINTI SUPARTO
MATRIC NUMBER	B17CS3033	A17CS0212

PREPARED FOR

Cik Suhaimi bin Yusof

(Section 02)

1. INTRODUCTION

In this assignment 2, we must apply viewing, shading, and texture effects to our objects using the C++ platform and OpenGL libraries. There are three simple objects (cube, pyramid, and icosahedron) and one special object (nut) that have been loaded from the.obj file. The programme runs one at a time, and there is no looping function to load the objects in the same window output multiple times. The programme menu will terminate after displaying the output of the user choices, as well as if the input is invalid.

2. OUTPUTS

From this project, the program succeeded in displaying the 3 objects (pyramid, cube and icosahedron) and a special object (nut) with shading and also the textures applications. Initially, a special object, nut where the file name is nut.obj can be displayed onto the windows, but after applying the menu for users to interact, the obj seems cannot be displayed but there is only a tiny bit of the application onto the object but cannot be seen clearly. Thus, we include a short video of the nut outputs in this assignment submission.

1. Basic Menu interactions which include the input from users. The system colors change to black background and white color for texts. The users will need to make the first choice which is the user needs to choose the shapes and second choice which is the user needs to choose what type of shading or applying texture according to the menu number. The input only accepts numbers from keyboards.

```
E:\UTM\UTM SEM8\SCSV4543 ADV COMP GRAPHICS\Assi...
=====MENU=====
Use ARROW KEYS to move in the scene.

=====

There will be 4 objects used in this project:
1. Pyramid
2. Cube
3. Icosahedron
4. Nut

=====Refer to this Key Functions=====

Press the following keys to change the object shading
(Flat/Gauraud) :
[q/Q] Pyramid
[w/W] Cube
[e/E] Icosahedron
[r/R] Nut

Press the following keys to change the object texture:
(Texture Mapping/Gauraud Texture) :
[a/A] Pyramid
[s/S] Cube
[d/D] Icosahedron
[f/F] Nut

Press the following keys to move the object:
[x/X] Rotate Object (x, -x)
[y/Y] Rotate Object (y, -y)
[z/Z] Rotate Object (z, -z)

Adjust Light percentages:
[i / o / p] + Diffuse Light(R, G, B)
[I / O / P] - Diffuse Light(R, G, B)
[j / k / l] + Specular Light(R, G, B)
[J / K / L] - Specular Light(R, G, B)
[b / n / m] + Ambient Light(R, G, B)
[B / N / M] - Ambient Light(R, G, B)

=====

Press 1 to start or 2 to exit =>
```

Figure 2.1 Basic Menu windows

2. All the objects are drawn using a grey material characteristic and a single light source illuminates the object. A 3D navigation system using `gluLookAt()` and

gluPerspective() functions has also been applied. The eye position does not change. Interaction: pressing the left, right, up and down keyboard button to navigate through the scene.

```

865 void processSpecialKeys(int key, int xx, int yy) {
866     float fraction = 0.1f;
867
868     switch (key) {
869     case GLUT_KEY_LEFT:
870         angle -= 0.01f;
871         lx = sin(angle);
872         lz = -cos(angle);
873         break;
874     case GLUT_KEY_RIGHT:
875         angle += 0.01f;
876         lx = sin(angle);
877         lz = -cos(angle);
878         break;
879     case GLUT_KEY_UP:
880         x += lx * fraction;
881         z += lz * fraction;
882         break;
883     case GLUT_KEY_DOWN:
884         x -= lx * fraction;
885         z -= lz * fraction;
886         break;
887     }
888 }
889

```

Figure 2.2 3D Navigation System (Arrow Keys)

There are also additional keys to change the object with the shading, mapping and texturing functions inside the *processNormalKeys* function. Table below shows the code of input keys for each function.

Table 2.1 Input key codes of functions onto the objects.

Function Keys	Input Key Codes
---------------	-----------------

Flat Shading	<pre> case 'q': glutDisplayFunc(drawFlatPyramid); glutIdleFunc(idle); break; case 'w': glutDisplayFunc(drawFlatCube); glutIdleFunc(idle); break; case 'e': glutDisplayFunc(drawFlatIcosahedron); glutIdleFunc(idle); break; case 'r': glutDisplayFunc(drawFlatNut); glutIdleFunc(idle); break; </pre>
Gouraud Shading	<pre> case 'Q': glutDisplayFunc(drawGouraudPyramid); glutIdleFunc(idle); break; case 'W': glutDisplayFunc(drawGouraudCube); glutIdleFunc(idle); break; case 'E': glutDisplayFunc(drawGouraudIcosahedron); glutIdleFunc(idle); break; case 'R': glutDisplayFunc(drawGouraudNut); glutIdleFunc(idle); break; </pre>

Texture Mapping	<pre> case 'a': glutDisplayFunc(drawFlatTexturePyramid); glutIdleFunc(idle); break; case 's': glutDisplayFunc(drawFlatTextureCube); glutIdleFunc(idle); break; case 'd': glutDisplayFunc(drawFlatTextureIcosahedron); glutIdleFunc(idle); break; case 'f': glutDisplayFunc(drawFlatTextureNut); glutIdleFunc(idle); break; </pre>
Gouraud Texture	<pre> case 'A': glutDisplayFunc(drawGouraudTexturePyramid); glutIdleFunc(idle); break; case 'S': glutDisplayFunc(drawGouraudTextureCube); glutIdleFunc(idle); break; case 'D': glutDisplayFunc(drawGouraudTextureIcosahedron); glutIdleFunc(idle); break; case 'F': glutDisplayFunc(drawGouraudTextureNut); glutIdleFunc(idle); break; </pre>

Rotate Object

```
case 'x':
    Xangle += 5.0;
    if (Xangle > 360.0) Xangle -= 360.0;
    glutPostRedisplay();
    break;
case 'X':
    Xangle -= 5.0;
    if (Xangle < 0.0) Xangle += 360.0;
    glutPostRedisplay();
    break;
case 'y':
    Yangle += 5.0;
    if (Yangle > 360.0) Yangle -= 360.0;
    glutPostRedisplay();
    break;
case 'Y':
    Yangle -= 5.0;
    if (Yangle < 0.0) Yangle += 360.0;
    glutPostRedisplay();
    break;
case 'z':
    Zangle += 5.0;
    if (Zangle > 360.0) Zangle -= 360.0;
    glutPostRedisplay();
    break;
case 'Z':
    Zangle -= 5.0;
    if (Zangle < 0.0) Zangle += 360.0;
    glutPostRedisplay();
    break;
```

Diffuse Light

```
case 'i':  
    light_diffuse[0] += 0.1;  
    glColor4fv(light_diffuse);  
    glutPostRedisplay();  
    break;  
case 'o':  
    light_diffuse[1] += 0.1;  
    glColor4fv(light_diffuse);  
    glutPostRedisplay();  
    break;  
case 'p':  
    light_diffuse[2] += 0.1;  
    glColor4fv(light_diffuse);  
    glutPostRedisplay();  
    break;  
case 'I':  
    light_diffuse[0] -= 0.1;  
    glColor4fv(light_diffuse);  
    glutPostRedisplay();  
    break;  
case 'O':  
    light_diffuse[1] -= 0.1;  
    glColor4fv(light_diffuse);  
    glutPostRedisplay();  
    break;  
case 'P':  
    light_diffuse[2] -= 0.1;  
    glColor4fv(light_diffuse);  
    glutPostRedisplay();  
    break;
```


Specular Light

```
case 'j':
    light_specular[0] += 0.1;
    glColor4fv(light_specular);
    glutPostRedisplay();
    break;
case 'k':
    light_specular[1] += 0.1;
    glColor4fv(light_specular);
    glutPostRedisplay();
    break;
case 'l':
    light_specular[2] += 0.1;
    glColor4fv(light_specular);
    glutPostRedisplay();
    break;
case 'J':
    light_specular[0] -= 0.1;
    glColor4fv(light_specular);
    glutPostRedisplay();
    break;
case 'K':
    light_specular[1] -= 0.1;
    glColor4fv(light_specular);
    glutPostRedisplay();
    break;
case 'L':
    light_specular[2] -= 0.1;
    glColor4fv(light_specular);
    glutPostRedisplay();
    break;
```

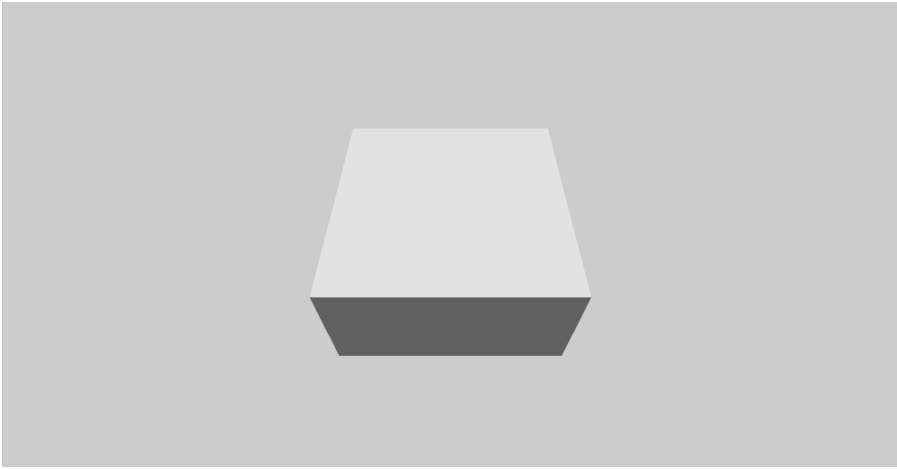


Ambient Light	<pre> case 'b': light_ambient[0] += 0.1; glColor4fv(light_ambient); glutPostRedisplay(); break; case 'n': light_ambient[1] += 0.1; glColor4fv(light_ambient); glutPostRedisplay(); break; case 'm': light_ambient[2] += 0.1; glColor4fv(light_ambient); glutPostRedisplay(); break; case 'B': light_ambient[0] -= 0.1; glColor4fv(light_ambient); glutPostRedisplay(); break; case 'N': light_ambient[1] -= 0.1; glColor4fv(light_ambient); glutPostRedisplay(); break; case 'M': light_ambient[2] -= 0.1; glColor4fv(light_ambient); glutPostRedisplay(); break; </pre>
---------------	--

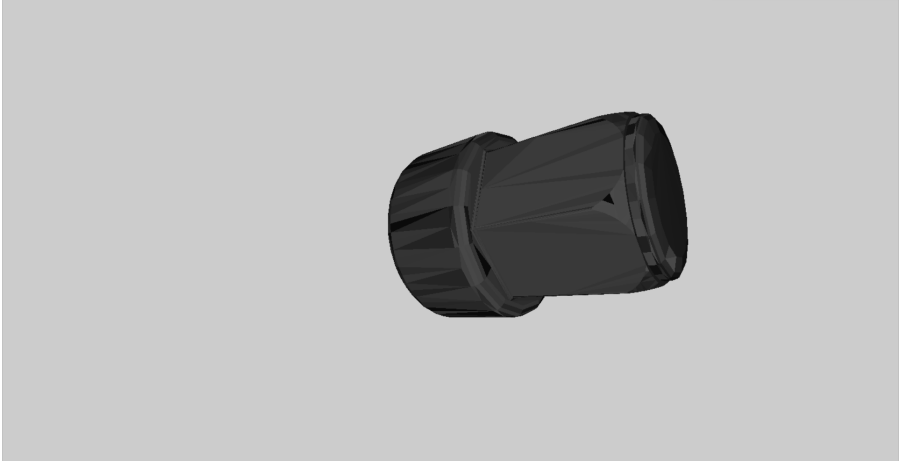
3. Apply shading models to the objects.

1) Flat Shading Model

Flat shading is enabled with `glShadeModel(GL_FLAT)`. The vertex normal is calculated by normalized cross product of any two sides. Table 2.1 below shows the flat shading outputs.

Table 2.2 Flat Shading applied onto the objects.

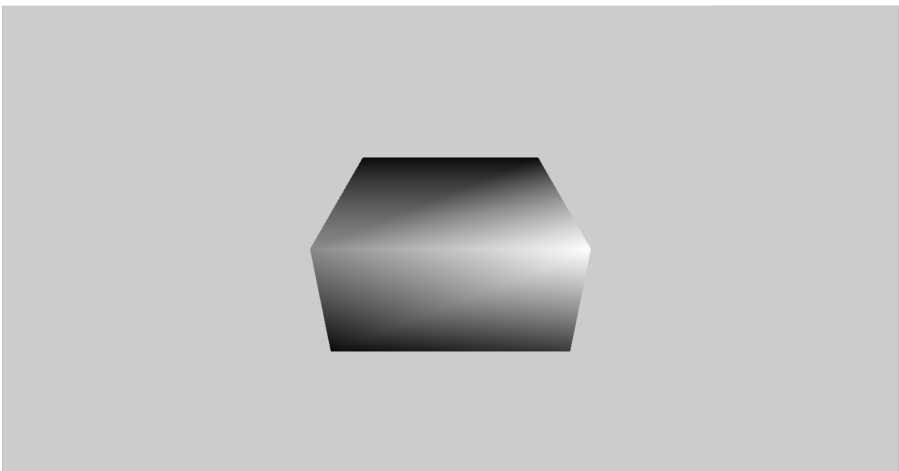
Object	Flat Shading Output
Cube	 <p>A 3D rendering of a cube with flat shading. The top face is light gray, the front face is medium gray, and the bottom face is dark gray. The cube is centered on a light gray background.</p>
Pyramid	 <p>A 3D rendering of a pyramid with flat shading. The front face is dark gray, the left face is medium gray, and the right face is light gray. The pyramid is centered on a light gray background.</p>
Icosahedron	 <p>A 3D rendering of an icosahedron with flat shading. The top face is light gray, the front face is medium gray, and the bottom face is dark gray. The icosahedron is centered on a light gray background.</p>

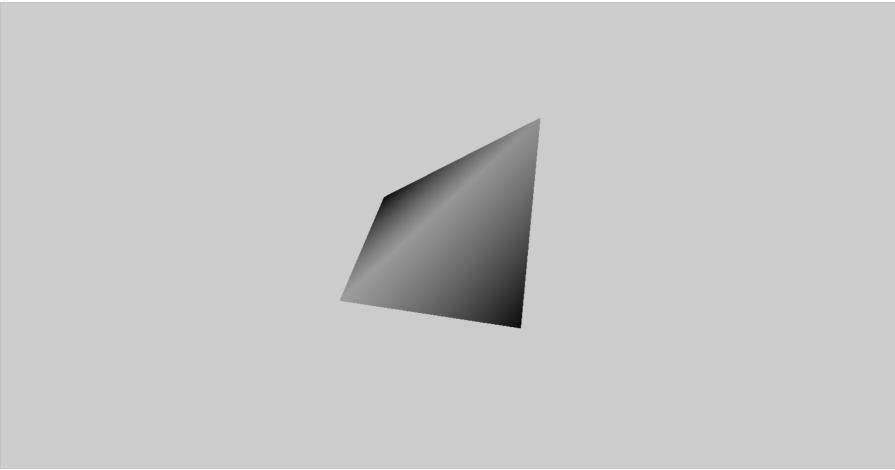
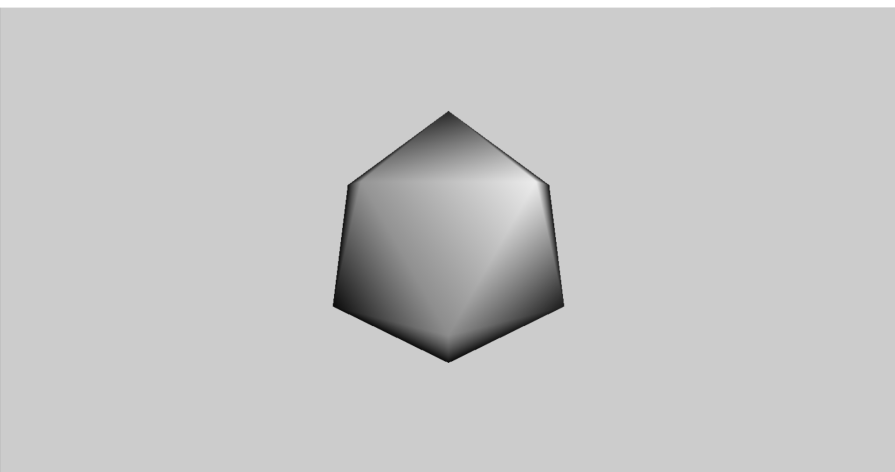
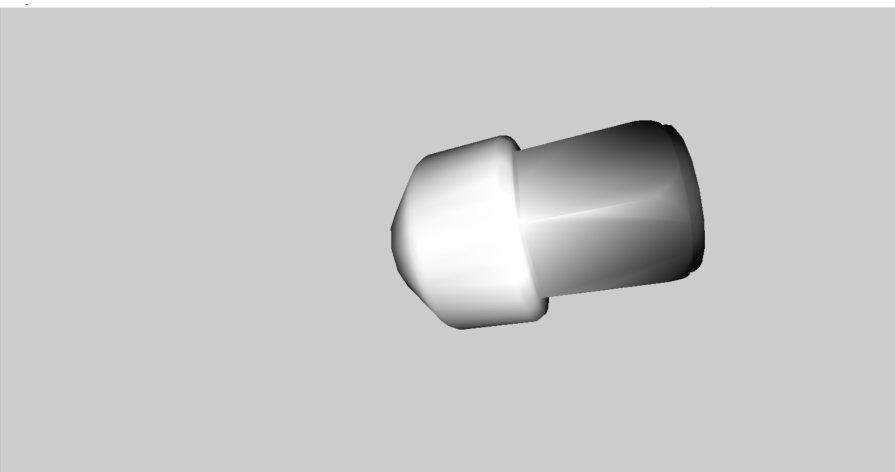
Nut	
-----	--

2) Gouraud Shading Model

Gouraud shading is enabled with `glShadeModel(GL_SMOOTH)`. The vertex normal is calculated by averaging all adjacent face normals, which is basically all faces that share a vertex.

Table 2.3 Gouraud Shading applied onto the objects.

Object	Gouraud Shading Output
Cube	

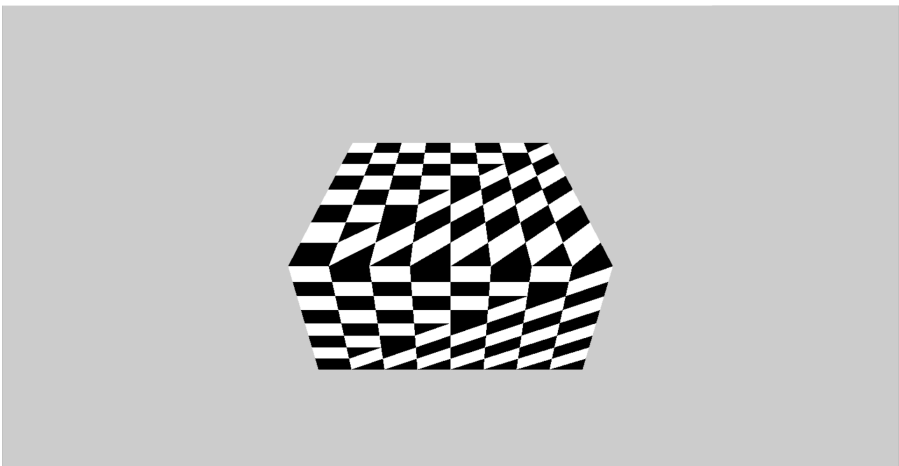
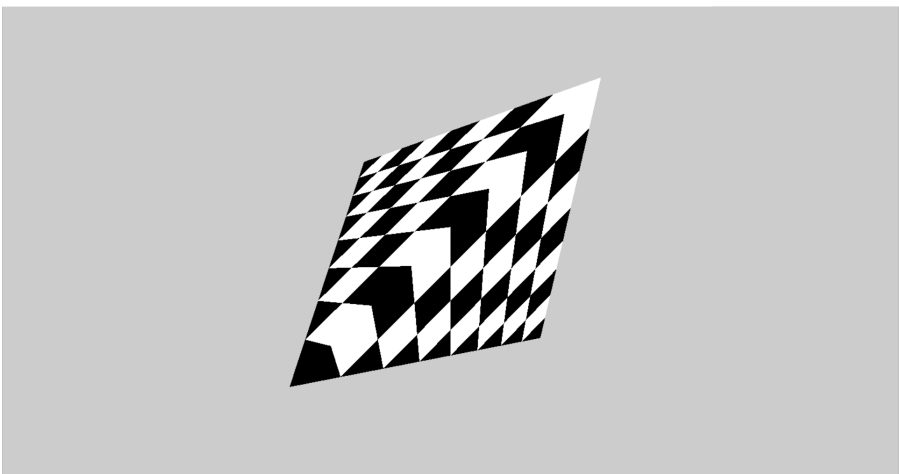
Pyramid	
Icosahedron	
Nut	

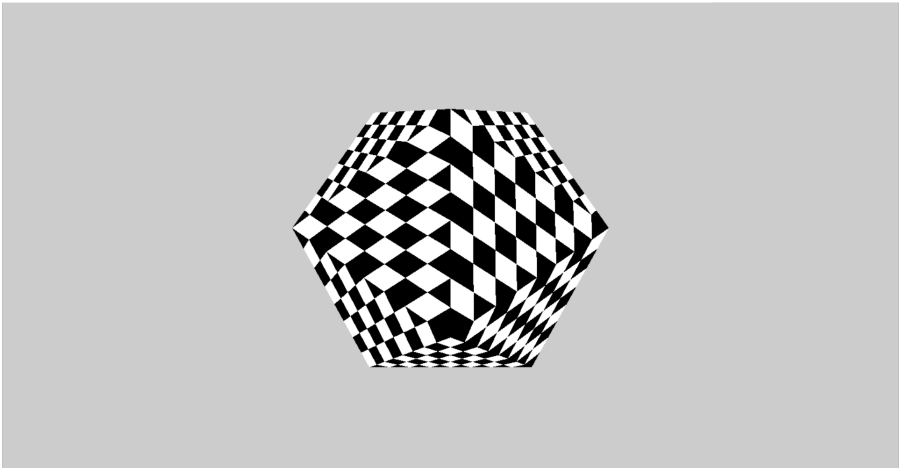
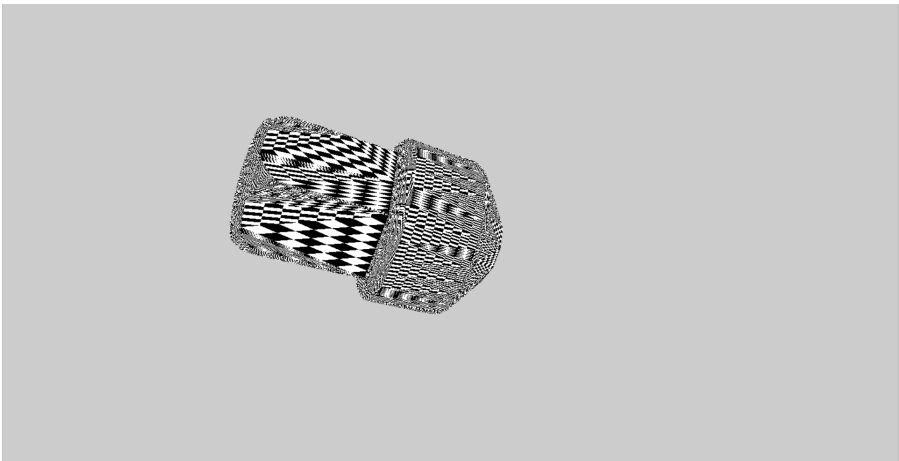
4. Apply texture mapping by assigning the created image to the selected object. The next function is applying the gouraud texture mapping by assigning the created image to the selected object.

1) Texture Mapping

This program uses `glBindTexture()` for creating and managing two textures. The texture is rendered as a checkerboard on the object. Initially, the object is drawn with texture coordinates based on the vertex's object coordinates.

Table 2.4 Texture Mapping applied onto the objects.

Object	Texture Mapping Output
Cube	
Pyramid	

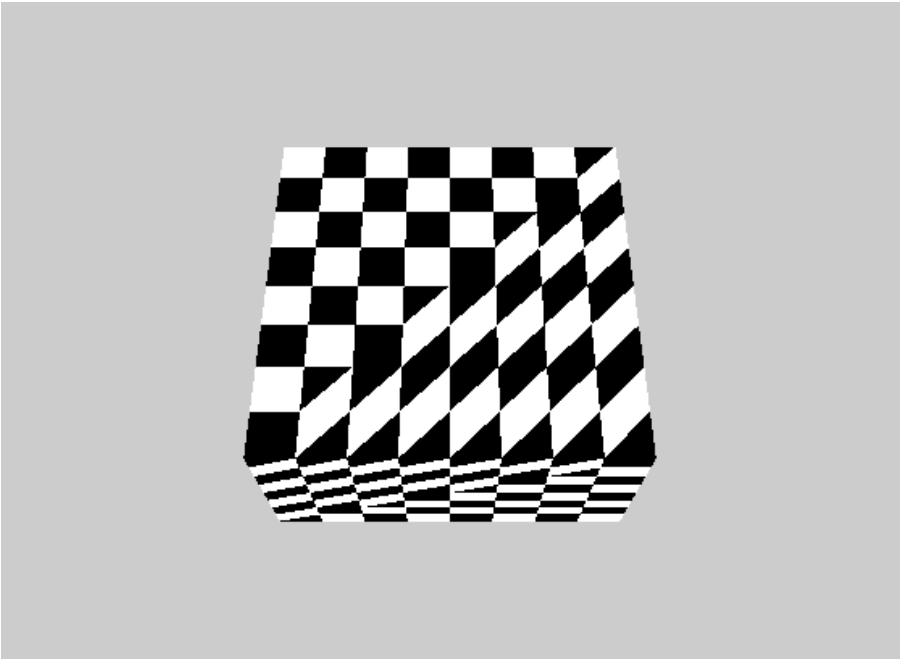
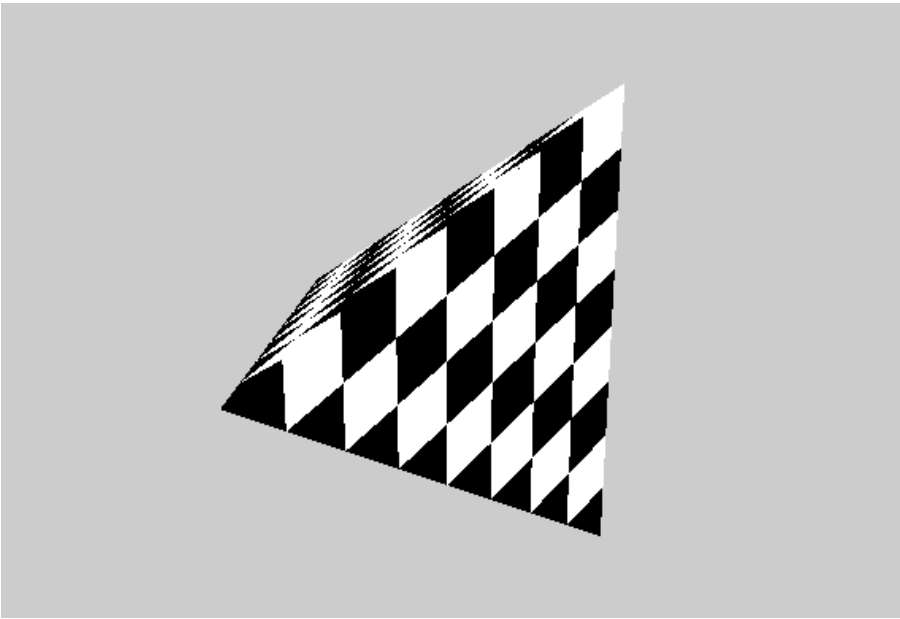
Icosahedron	
Nut	

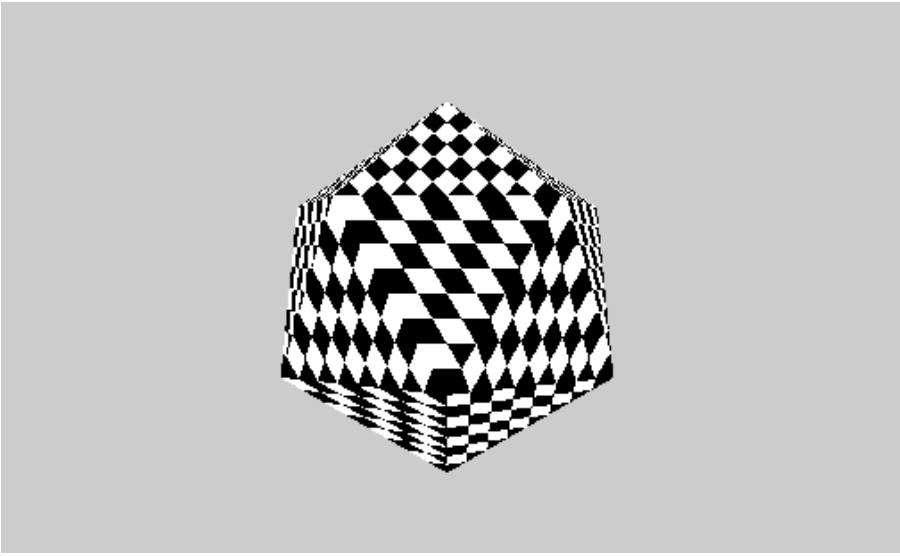
2) Gouraud Texture Mapping

Applying the Gouraud texture mapping shows the object output clearly with its shapes, different when applying the only texture mapping. For example, it shows clearly that the shape of the pyramid is more to pyramid shape after the gouraud texture application is applied.

Table 2.5 Gouraud Texture applied onto the objects.

Object	Gouraud Texture Mapping
--------	-------------------------

Cube	
Pyramid	

Icosahedron	
Nut	