Batch: D-2 **Roll No.:** 16010122151
Experiment No. 08

TITLE: Write an OpenGL program to implement Shadow Mapping.

AIM:

Write an OpenGL program to implement Shadow Mapping.
Create 3D object and demonstrate the shadow of same object.

Expected OUTCOME of Experiment:

Books/ Journals/ Websites referred:

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

Algorithm/ Pseudocode for each process:

1. Render the scene using the light as the camera and perform z-buffering
2. Generate a light z buffer
3. Render the scene using the regular camera, perform z-buffering

Implementation details:

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
from OpenGL.GL.shaders import *
from OpenGL.GL.framebufferobjects import *
import math
class Camera:
 def __init__(self):
 self.rotx, self.roty = math.pi/4, math.pi/4
 self.distance = 100
 self.moving = False
 self.ex, self.ey = 0, 0
 self.size = (800, 600)
 def load_matrices(self):
 glViewport(0, 0, *self.size)
 y = math.cos(self.roty) * self.distance
 x = math.sin(self.roty) * math.cos(self.rotx) * self.distance
 z = math.sin(self.roty) * math.sin(self.rotx) * self.distance
 glMatrixMode(GL_PROJECTION)
 glLoadIdentity()
 gluPerspective(45.0, self.size[0]/float(self.size[1]), 1, 1000)
 glMatrixMode(GL_MODELVIEW)
 glLoadIdentity()
 gluLookAt(x,y,z, 0,0,0, 0,1,0)
 def on_mouse_button (self, b, s, x, y):
 self.moving = not s
 self.ex, self.ey = x, y
 if b in [3, 4]:
 dz = (1 if b == 3 else -1)
 self.distance += self.distance/15.0 * dz;
 def on_mouse_move(self, x, y, z = 0):
 if self.moving:
 self.rotx += (x-self.ex) / 300.0
 self.roty += -(y-self.ey) / 300.0
self.ex, self.ey = x, y
 def set_size(self, w, h):
 self.size = w, h
class Shader():
```

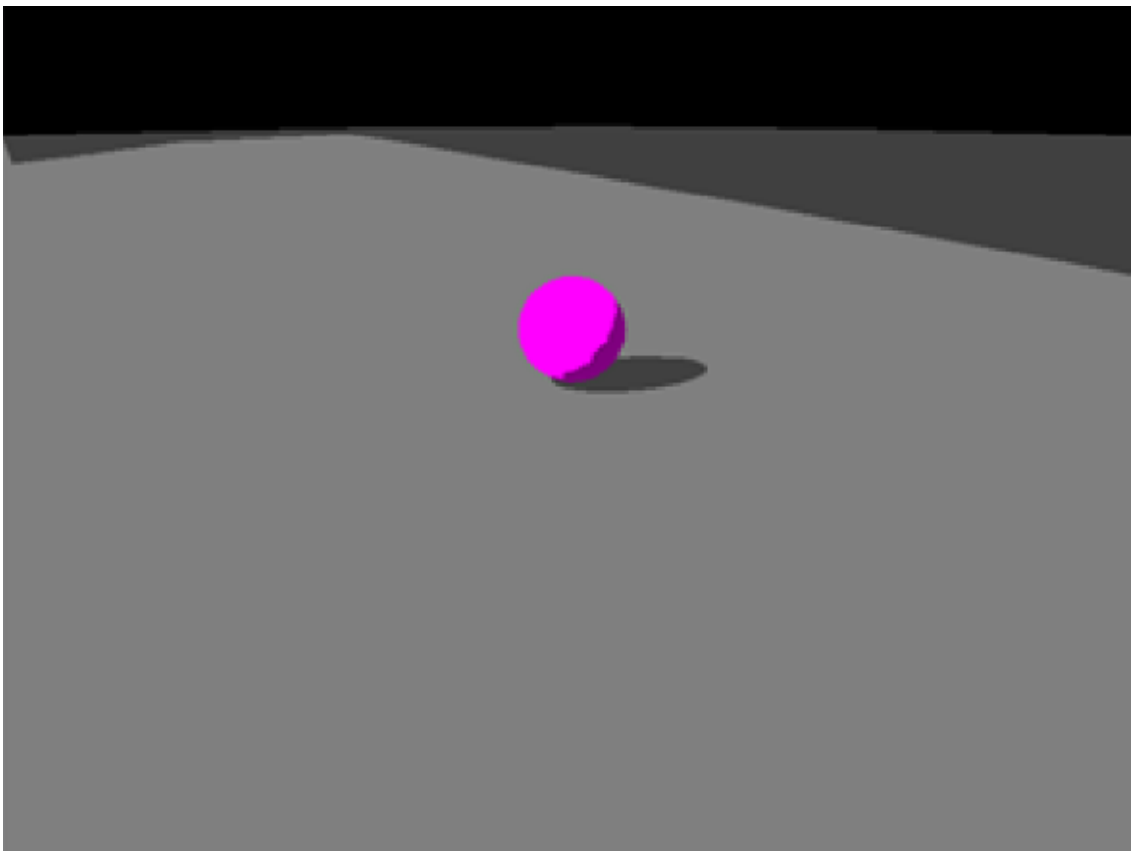
```
def __init__(self):
self.is_built = False
self.uniforms = {}
def build(self):
self.program = compileProgram(
compileShader('''
uniform mat4 camMatrix;
uniform mat4 shadowMatrix;
varying vec4 depthProjection;
uniform bool useShadow;
void main() {
gl_Position = camMatrix * gl_ModelViewMatrix * gl_Vertex;
depthProjection = shadowMatrix * gl_ModelViewMatrix * gl_Vertex;
gl_FrontColor = gl_Color;
}
''',GL_VERTEX_SHADER),
compileShader('''
varying vec4 depthProjection;
uniform sampler2D shadowMap;
uniform bool useShadow;
void main () {
float shadow = 1.0;
if (useShadow) {
vec4 shadowCoord = depthProjection / depthProjection.w ;
shadowCoord.z -= 0.0002;
float distanceFromLight = texture2D(shadowMap, shadowCoord.st).z;
if (depthProjection .w > 0.0)
shadow = distanceFromLight < shadowCoord.z ? 0.5 : 1.0 ;
}
gl_FragColor = shadow * gl_Color;
}
''',GL_FRAGMENT_SHADER),)
self.is_built = True
self.uniforms['camMatrix'] = glGetUniformLocation(self.program, 'camMatrix')
self.uniforms['shadowMatrix'] = glGetUniformLocation(self.program,
'shadowMatrix')
self.uniforms['shadowMap'] = glGetUniformLocation(self.program, 'shadowMap')
self.uniforms['useShadow'] = glGetUniformLocation(self.program, 'useShadow')
print(self.uniforms)
def use(self):
if not self.is_built:
self.build()
glUseProgram(self.program)
class Test:
def __init__(self):
glutInit(sys.argv)
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_ALPHA |
GLUT_DEPTH)
glutInitWindowSize(800, 600)
```

```
# glutInitWindowPosition(1120/2, 100)
self.window = glutCreateWindow("Shadow Mapping")
self.cam = Camera()
self.light = Camera()
self.cam.set_size(800, 600)
self.light.set_size(2048, 2048)
self.light.distance = 100
self.shader = Shader()
self.initialized = False
def setup(self):
self.initialized = True
glClearColor(0,0,0,1.0);
glDepthFunc(GL_LESS)
glEnable(GL_DEPTH_TEST)
self.fbo = glGenFramebuffers(1);
self.shadowTexture = glGenTextures(1)
glBindFramebuffer(GL_FRAMEBUFFER, self.fbo)
w, h = self.light.size
glActiveTexture(GL_TEXTURE5)
glBindTexture(GL_TEXTURE_2D, self.shadowTexture)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP );
glTexImage2D( GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, w, h, 0,
GL_DEPTH_COMPONENT, GL_UNSIGNED_INT, None)
glDrawBuffer(GL_NONE)
glReadBuffer(GL_NONE)
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_TEXTURE_2D, self.fbo, 0)
FBOstatus = glCheckFramebufferStatus(GL_FRAMEBUFFER)
if FBOstatus != GL_FRAMEBUFFER_COMPLETE:
print ("GL_FRAMEBUFFER_COMPLETE_EXT failed, CANNOT use
FBO\n");
glBindFramebuffer(GL_FRAMEBUFFER, 0)
#glActiveTexture(GL_TEXTURE0)
def draw(self):
glPushMatrix()
glTranslate(0, 10 ,0)
glColor4f(1, 0, 1, 0)
glutSolidSphere(5, 14, 12)
glPopMatrix()
glPushMatrix()
glColor4f(0.5, 0.5, .5, 1)
glScale(100, 1, 100)
```

```
glutSolidSphere(5, 14, 12)
glPopMatrix()
def apply_camera(self, cam):
cam.load_matrices()
model_view = glGetDoublev(GL_MODELVIEW_MATRIX);
projection = glGetDoublev(GL_PROJECTION_MATRIX);
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
glMultMatrixd(projection)
glMultMatrixd(model_view)
glUniformMatrix4fv(self.shader.uniforms['camMatrix'], 1, False,
glGetFloatv(GL_MODELVIEW_MATRIX))
glLoadIdentity()
def shadow_pass(self):
glUniform1i(self.shader.uniforms['useShadow'], 0)
glBindFramebuffer(GL_FRAMEBUFFER, self.fbo)
glClear(GL_DEPTH_BUFFER_BIT)
glCullFace(GL_FRONT)
self.apply_camera(self.light)
self.draw()
glBindFramebuffer(GL_FRAMEBUFFER, 0)
def final_pass(self):
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
self.light.load_matrices()
model_view = glGetDoublev(GL_MODELVIEW_MATRIX);
projection = glGetDoublev(GL_PROJECTION_MATRIX);
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
bias = [ 0.5, 0.0, 0.0, 0.0,
0.0, 0.5, 0.0, 0.0,
0.0, 0.0, 0.5, 0.0,
0.5, 0.5, 0.5, 1.0]
glLoadMatrixd(bias)
glMultMatrixd(projection)
glMultMatrixd(model_view)
glUniformMatrix4fv(self.shader.uniforms['shadowMatrix'], 1, False,
glGetFloatv(GL_MODELVIEW_MATRIX))
glActiveTexture(GL_TEXTURE5)
glBindTexture(GL_TEXTURE_2D, self.shadowTexture)
glUniform1i(self.shader.uniforms['shadowMap'], 5)
glUniform1i(self.shader.uniforms['useShadow'], 1);
self.apply_camera(self.cam)
glLoadIdentity()
glCullFace(GL_BACK)
self.draw()
def render(self):
if not self.initialized: self.setup()
self.shader.use()
self.shadow_pass()
```

```
 self.final_pass()
 glutSwapBuffers()
 def mouse_move(self, *args):
 self.cam.on_mouse_move(*args)
 self.light.on_mouse_move(*args)
 def mouse_button(self, b, *args):
 if b==0:
 self.light.on_mouse_button(b, *args)
else:
 self.cam.on_mouse_button(b, *args)
 def main(self):
 glutDisplayFunc(self.render)
 glutIdleFunc(self.render)
 glutMouseFunc(self.mouse_button)
 glutMotionFunc(self.mouse_move)
 glutReshapeFunc(self.cam.set_size)
 #self.setup()
 glutMainLoop()
if __name__ == '__main__':
 test = Test()
 test.main(
```

Output(s) (Screen Shot):



Conclusion and discussion:

This experiment clarified the concept and implementation of Shadow Mapping in PyOpenGL.

Post lab question
Write a program to demonstrate shadow for two objects.
Write to program to implement various curves (at least two - three types of curve)

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
from OpenGL.GL.shaders import *
from OpenGL.GL.framebufferobjects import *
import math
class Camera:
 def __init__(self):
 self.rotx, self.roty = math.pi/4, math.pi/4
 self.distance = 100
 self.moving = False
 self.ex, self.ey = 0, 0
 self.size = (800, 600)
 def load_matrices(self):
 glViewport(0, 0, *self.size)
 y = math.cos(self.roty) * self.distance
 x = math.sin(self.roty) * math.cos(self.rotx) * self.distance
 z = math.sin(self.roty) * math.sin(self.rotx) * self.distance
 glMatrixMode(GL_PROJECTION)
 glLoadIdentity()
 gluPerspective(45.0, self.size[0]/float(self.size[1]), 1, 1000)
 glMatrixMode(GL_MODELVIEW)
 glLoadIdentity()
 gluLookAt(x,y,z, 0,0,0, 0,1,0)
 def on_mouse_button (self, b, s, x, y):
 self.moving = not s
 self.ex, self.ey = x, y
 if b in [3, 4]:
 dz = (1 if b == 3 else -1)
 self.distance += self.distance/15.0 * dz;
 def on_mouse_move(self, x, y, z = 0):
 if self.moving:
 self.rotx += (x-self.ex) / 300.0
 self.roty += -(y-self.ey) / 300.0
 self.ex, self.ey = x, y
```

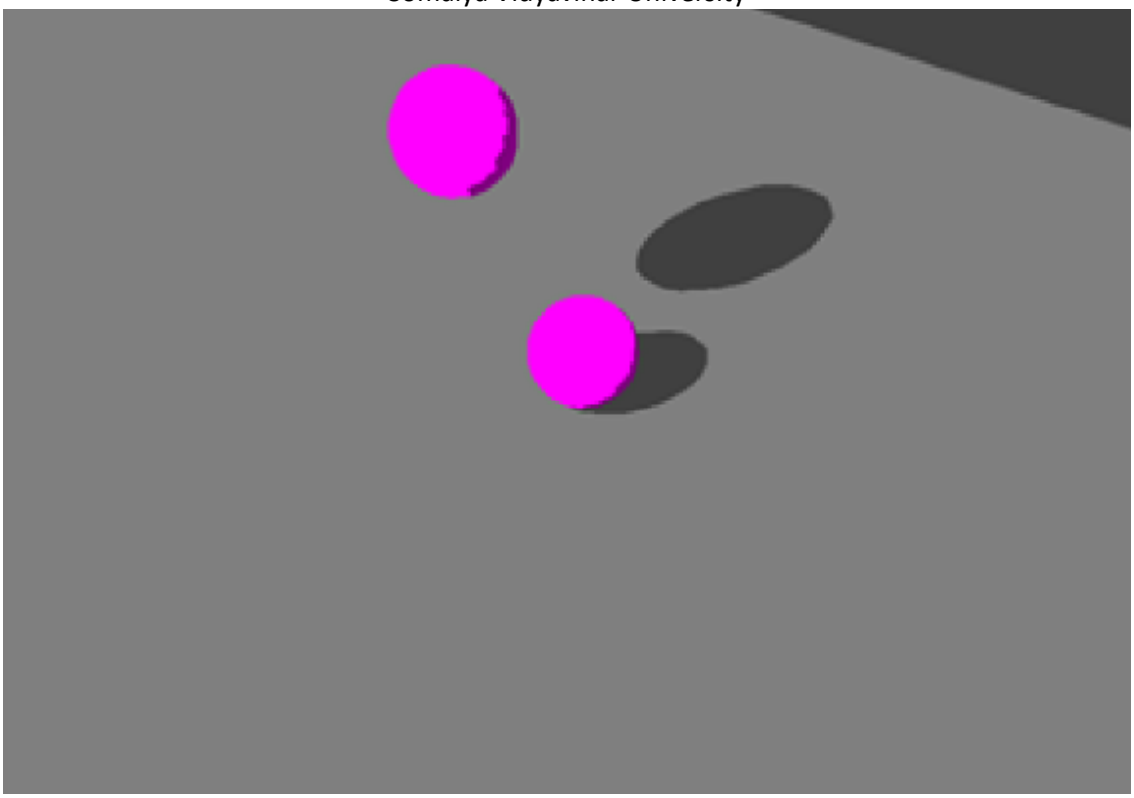
```
 def set_size(self, w, h):
self.size = w, h
class Shader():
 def __init__(self):
 self.is_built = False
 self.uniforms = {}
 def build(self):
 self.program = compileProgram(
 compileShader('''
 uniform mat4 camMatrix;
 uniform mat4 shadowMatrix;
 varying vec4 depthProjection;
 uniform bool useShadow;
 void main() {
 gl_Position = camMatrix * gl_ModelViewMatrix * gl_Vertex;
 depthProjection = shadowMatrix * gl_ModelViewMatrix * gl_Vertex;
 gl_FrontColor = gl_Color;
 }
 ''',GL_VERTEX_SHADER),
 compileShader('''
 varying vec4 depthProjection;
 uniform sampler2D shadowMap;
 uniform bool useShadow;
 void main () {
 float shadow = 1.0;
 if (useShadow) {
 vec4 shadowCoord = depthProjection / depthProjection.w ;
 shadowCoord.z -= 0.0002;
 float distanceFromLight = texture2D(shadowMap, shadowCoord.st).z;
 if (depthProjection .w > 0.0)
 shadow = distanceFromLight < shadowCoord.z ? 0.5 : 1.0 ;
 }
 gl_FragColor = shadow * gl_Color;
 }
 ''',GL_FRAGMENT_SHADER),)
 self.is_built = True
 self.uniforms['camMatrix'] = glGetUniformLocation(self.program, 'camMatrix')
 self.uniforms['shadowMatrix'] = glGetUniformLocation(self.program,
'shadowMatrix')
 self.uniforms['shadowMap'] = glGetUniformLocation(self.program, 'shadowMap')
 self.uniforms['useShadow'] = glGetUniformLocation(self.program, 'useShadow')
 print(self.uniforms)
def use(self):
 if not self.is_built:
 self.build()
 glUseProgram(self.program)
class Test:
 def __init__(self):
 glutInit(sys.argv)
```

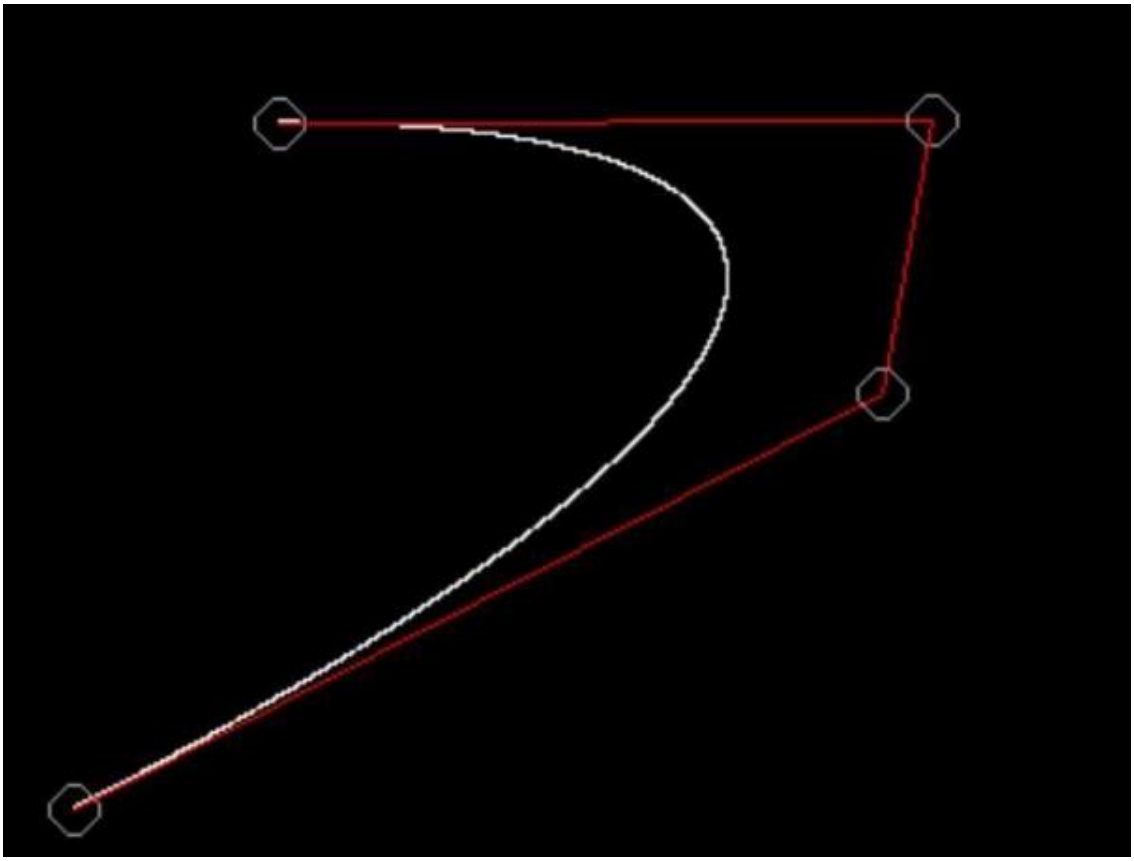
```
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_ALPHA |
GLUT_DEPTH)
glutInitWindowSize(800, 600)
# glutInitWindowPosition(1120/2, 100)
self.window = glutCreateWindow("Shadow Test")
self.cam = Camera()
self.light = Camera()
self.cam.set_size(800, 600)
self.light.set_size(2048, 2048)
self.light.distance = 100
self.shader = Shader()
self.initialized = False
def setup(self):
self.initialized = True
glClearColor(0,0,0,1.0);
glDepthFunc(GL_LESS)
glEnable(GL_DEPTH_TEST)
self.fbo = glGenFramebuffers(1);
self.shadowTexture = glGenTextures(1)
glBindFramebuffer(GL_FRAMEBUFFER, self.fbo)
w, h = self.light.size
glActiveTexture(GL_TEXTURE5)
glBindTexture(GL_TEXTURE_2D, self.shadowTexture)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP );
glTexImage2D( GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, w, h, 0,
GL_DEPTH_COMPONENT, GL_UNSIGNED_INT, None)
glDrawBuffer(GL_NONE)
glReadBuffer(GL_NONE)
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_TEXTURE_2D, self.fbo, 0)
FBOstatus = glCheckFramebufferStatus(GL_FRAMEBUFFER)
if FBOstatus != GL_FRAMEBUFFER_COMPLETE:
print ("GL_FRAMEBUFFER_COMPLETE_EXT failed, CANNOT use
FBO\n");
glBindFramebuffer(GL_FRAMEBUFFER, 0)
#glActiveTexture(GL_TEXTURE0)
def draw(self):
glPushMatrix()
glTranslate(0, 10 ,0)
glColor4f(1, 0, 1, 0)
glutSolidSphere(5, 14, 12)
glPopMatrix()
```

```
 glPushMatrix()
 glTranslate(0, 30 ,10)
 glColor4f(1, 0, 1, 0)
 glutSolidSphere(5, 14, 12)
 glPopMatrix()
a
 glPushMatrix()
 glColor4f(0.5, 0.5, .5, 1)
 glScale(100, 1, 100)
 glutSolidSphere(5, 14, 12)
 glPopMatrix()
 def apply_camera(self, cam):
 cam.load_matrices()
 model_view = glGetDoublev(GL_MODELVIEW_MATRIX);
 projection = glGetDoublev(GL_PROJECTION_MATRIX);
 glMatrixMode(GL_MODELVIEW)
 glLoadIdentity()
 glMultMatrixd(projection)
 glMultMatrixd(model_view)
 glUniformMatrix4fv(self.shader.uniforms['camMatrix'], 1, False,
glGetFloatv(GL_MODELVIEW_MATRIX))
 glLoadIdentity()
 def shadow_pass(self):
glUniform1i(self.shader.uniforms['useShadow'], 0)
 glBindFramebuffer(GL_FRAMEBUFFER, self.fbo)
 glClear(GL_DEPTH_BUFFER_BIT)
 glCullFace(GL_FRONT)
 self.apply_camera(self.light)
 self.draw()
 glBindFramebuffer(GL_FRAMEBUFFER, 0)
 def final_pass(self):
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
 self.light.load_matrices()
 model_view = glGetDoublev(GL_MODELVIEW_MATRIX);
 projection = glGetDoublev(GL_PROJECTION_MATRIX);
 glMatrixMode(GL_MODELVIEW)
 glLoadIdentity()
 bias = [ 0.5, 0.0, 0.0, 0.0,
 0.0, 0.5, 0.0, 0.0,
 0.0, 0.0, 0.5, 0.0,
 0.5, 0.5, 0.5, 1.0]
 glLoadMatrixd(bias)
 glMultMatrixd(projection)
 glMultMatrixd(model_view)
 glUniformMatrix4fv(self.shader.uniforms['shadowMatrix'], 1, False,
glGetFloatv(GL_MODELVIEW_MATRIX))
 glActiveTexture(GL_TEXTURE5)
 glBindTexture(GL_TEXTURE_2D, self.shadowTexture)
 glUniform1i(self.shader.uniforms['shadowMap'], 5)
```

```
 glUniform1i(self.shader.uniforms['useShadow'], 1);
 self.apply_camera(self.cam)
 glLoadIdentity()
 glCullFace(GL_BACK)
 self.draw()
 def render(self):
 if not self.initialized: self.setup()
 self.shader.use()
 self.shadow_pass()
 self.final_pass()
 glutSwapBuffers()
 def mouse_move(self, *args):
 self.cam.on_mouse_move(*args)
 self.light.on_mouse_move(*args)
def mouse_button(self, b, *args):
 if b==0:
 self.light.on_mouse_button(b, *args)
 else:
 self.cam.on_mouse_button(b, *args)
 def main(self):
 glutDisplayFunc(self.render)
 glutIdleFunc(self.render)
 glutMouseFunc(self.mouse_button)
 glutMotionFunc(self.mouse_move)
 glutReshapeFunc(self.cam.set_size)
 #self.setup()
 glutMainLoop()
if __name__ == '__main__':
 test = Test()
 test.main()
```

Output:





Date: 18-10-2024

Signature of faculty in-charge