ПРАКТИЧЕСКАЯ РАБОТА №3

по дисциплине

«Компьютерная графика»

**Вариант 1**

|  |  |
| --- | --- |
| Выполнил  студент гр. 5130904/10101 | Абраамян А. М. |
| Преподаватель | Леонтьева Т.В. |

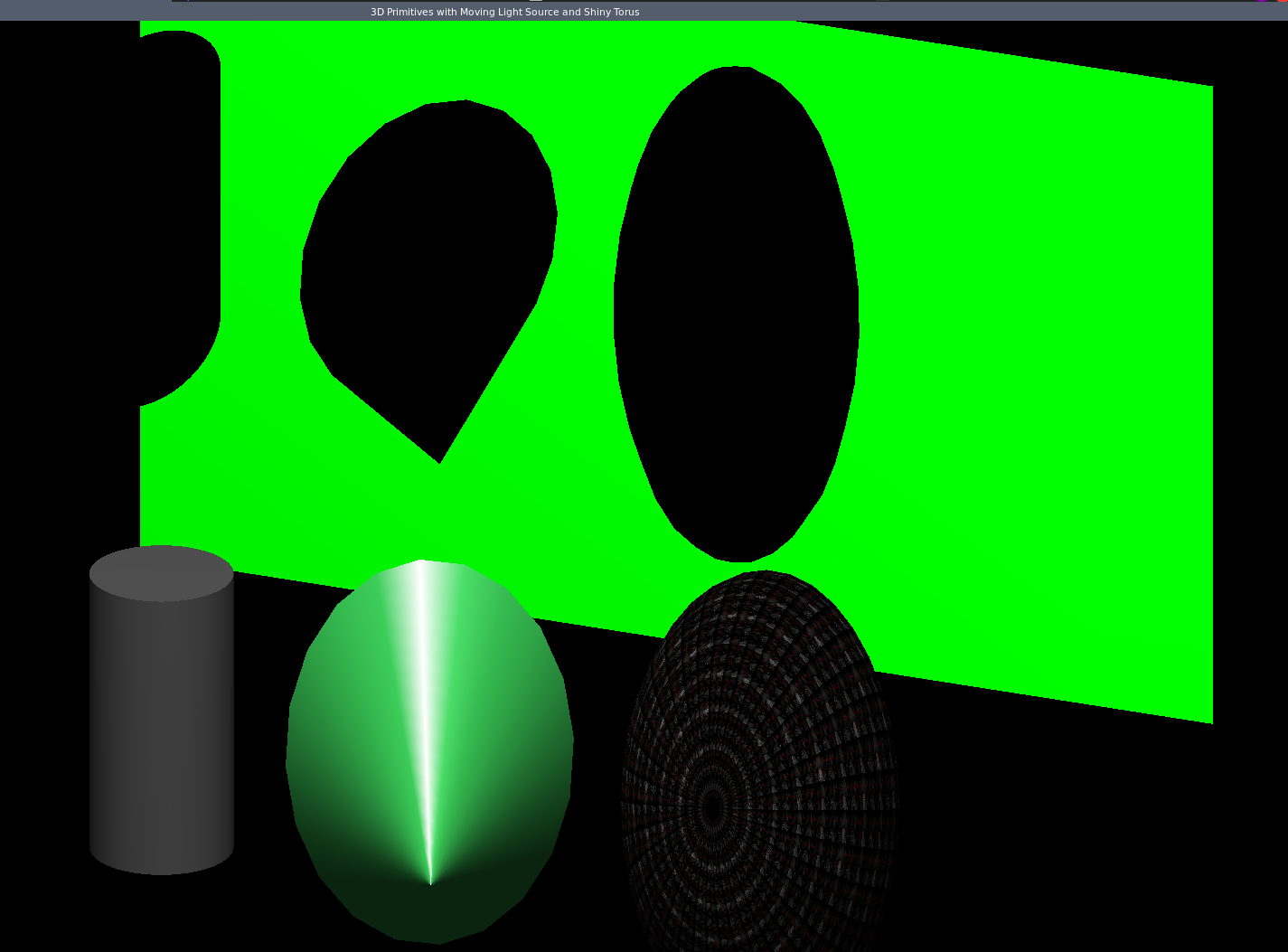
# Цель работы

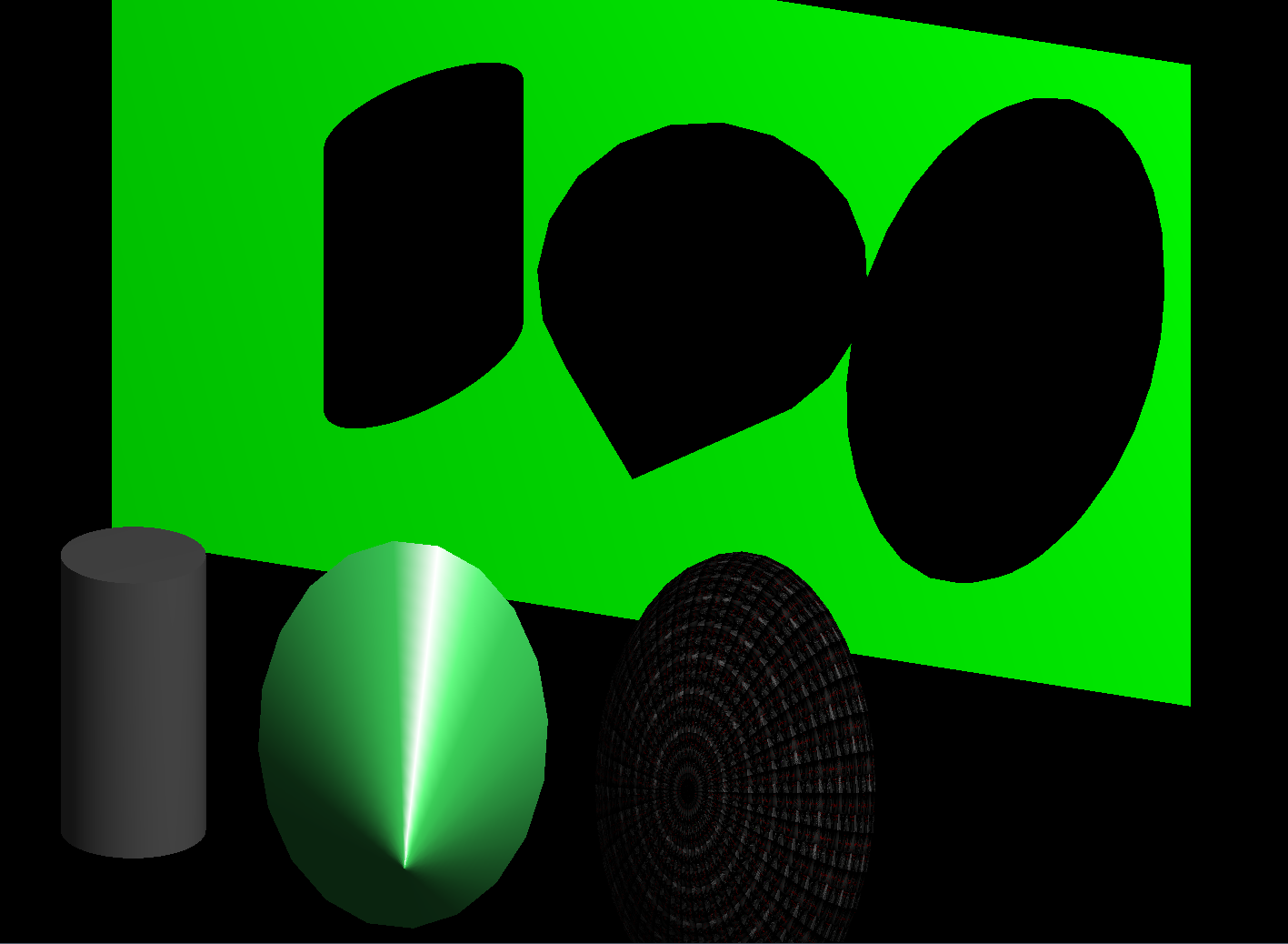
Целью работы является расчёт и визуализация динамических теней от объектов, отбрасываемых на плоскость (по желанию, также на другие объекты). Требуется разработать программу, демонстрирующую тот же набор из трех предметов, который был выбран для вашего варианта задания №2, с сохранением их свойств материалов. Эти предметы следует расположить на плоскости или над плоскостью. Дополнительно сами объекты и плоскость можно текстурировать. В сцене должен присутствовать один источник освещения, меняющий местоположение, при этом требуется рассчитывать и отображать тени, которые объекты отбрасывают на плоскость, на/над которой они находятся.

# Порядок выполнения

1. Были использованы фигуры для отражения тени: **конус, циллиндр, текстурированная сфера**.
2. В сцене должен присутствовать один источник освещения, меняющий местоположение, при этом требуется рассчитывать и отображать тени, которые объекты отбрасывают на плоскость, на/над которой они находятся.

# Результат работы





# Код программы

|  |
| --- |
| from OpenGL.GL import \*  from OpenGL.GLU import \*  from OpenGL.GLUT import \*  import sys  import math  import numpy as np  from PIL import Image  # Параметры освещения  light\_angle = 0  light\_position = [0.0, 7.6, 10.0, 1.0] # Источник света над объектами  light\_color = [1.0, 1.0, 1.0, 1.0]  light\_intensity = 1  texture\_id = None  def load\_image(image\_path):  image = Image.open(image\_path)  return image.transpose(Image.FLIP\_TOP\_BOTTOM)  def initialize\_texture(image):  texture\_data = np.array(image.convert("RGBA"), np.uint8)  glBindTexture(GL\_TEXTURE\_2D, texture\_id)  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT)  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT)  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR)  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR)  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, image.width, image.height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, texture\_data)  glBindTexture(GL\_TEXTURE\_2D, 0)  def load\_texture(image\_path):  global texture\_id  texture\_id = glGenTextures(1)  image = load\_image(image\_path)  initialize\_texture(image)  def draw\_cylinder\_surface(radius, height, num\_segments):  glBegin(GL\_QUAD\_STRIP)  for i in range(num\_segments + 1):  theta = 2.0 \* math.pi \* i / num\_segments  x = radius \* math.cos(theta)  z = radius \* math.sin(theta)  glNormal3f(x, 0.0, z)  glVertex3f(x, height / 2, z)  glVertex3f(x, -height / 2, z)  glEnd()  def draw\_cylinder\_cap(radius, height, num\_segments, is\_top):  glBegin(GL\_TRIANGLE\_FAN)  glNormal3f(0.0, 1.0 if is\_top else -1.0, 0.0)  glVertex3f(0.0, height / 2 if is\_top else -height / 2, 0.0)  for i in range(num\_segments + 1):  theta = 2.0 \* math.pi \* i / num\_segments  x = radius \* math.cos(theta)  z = radius \* math.sin(theta)  glVertex3f(x, height / 2 if is\_top else -height / 2, z)  glEnd()  def draw\_smooth\_cylinder(radius, height, num\_segments):  draw\_cylinder\_surface(radius, height, num\_segments)  draw\_cylinder\_cap(radius, height, num\_segments, True)  draw\_cylinder\_cap(radius, height, num\_segments, False)  def draw\_textured\_sphere(texture\_id, radius, num\_segments\_lat, num\_segments\_long):  glEnable(GL\_TEXTURE\_2D)  glBindTexture(GL\_TEXTURE\_2D, texture\_id)  glBegin(GL\_QUADS)  for i in range(-90, 90, num\_segments\_lat):  for j in range(0, 360, num\_segments\_long):  theta1 = math.radians(i)  theta2 = math.radians(i + num\_segments\_lat)  phi1 = math.radians(j)  phi2 = math.radians(j + num\_segments\_long)  v1 = [math.sin(theta1) \* math.cos(phi1), math.cos(theta1), math.sin(theta1) \* math.sin(phi1)]  v2 = [math.sin(theta2) \* math.cos(phi1), math.cos(theta2), math.sin(theta2) \* math.sin(phi1)]  v3 = [math.sin(theta2) \* math.cos(phi2), math.cos(theta2), math.sin(theta2) \* math.sin(phi2)]  v4 = [math.sin(theta1) \* math.cos(phi2), math.cos(theta1), math.sin(theta1) \* math.sin(phi2)]  glNormal3fv(v1)  glTexCoord2f(0.0, 0.0)  glVertex3fv(v1)  glNormal3fv(v2)  glTexCoord2f(0.6, 0.0)  glVertex3fv(v2)  glNormal3fv(v3)  glTexCoord2f(0.6, 0.6)  glVertex3fv(v3)  glNormal3fv(v4)  glTexCoord2f(0.0, 0.6)  glVertex3fv(v4)  glEnd()  glBindTexture(GL\_TEXTURE\_2D, 0)  glDisable(GL\_TEXTURE\_2D)  def draw\_plane():  glBegin(GL\_QUADS)  glNormal3f(0.0, 0.0, 1.0)  glColor3f(0, 1, 0)  glVertex3f(-3.7, -1.0, -1.6)  glVertex3f(4.3, -1.0, -1.6)  glVertex3f(4.3, 1.8, -1.6)  glVertex3f(-3.7, 1.8, -1.6)  glEnd()  def init\_lighting():  glEnable(GL\_LIGHTING)  glEnable(GL\_LIGHT0)  glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_color)  glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_color)  glEnable(GL\_COLOR\_MATERIAL)  glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE)  def init\_opengl():  glClearColor(0.0, 0.0, 0.0, 1.0)  glMatrixMode(GL\_PROJECTION)  glOrtho(-6.0, 6.0, -2.0, 2.0, 0.1, 50.0)  glMatrixMode(GL\_MODELVIEW)  init\_lighting()  glEnable(GL\_BLEND)  glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA)  load\_texture("/home/hryapusek/work/7th\_sem/opengl/polytech-opengl/lab3/textures/cone\_texture.png")  moving\_right = True  def update\_light\_position():  global light\_position, moving\_right  # Плавное движение источника света  if moving\_right:  light\_position[0] += 0.1 # Двигаемся вправо  if light\_position[0] >= 14.0: # Если достигли правой границы  moving\_right = False # Меняем направление на левое  else:  light\_position[0] -= 0.05 # Двигаемся влево  if light\_position[0] <= -9.0: # Если достигли левой границы  moving\_right = True # Меняем направление на правое  # Обновляем позицию света  glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position)  def set\_lighting\_intensity():  intensity = [c \* light\_intensity for c in light\_color]  glLightfv(GL\_LIGHT0, GL\_DIFFUSE, intensity + [1.0])  def configure\_material\_shiny():  glMaterialfv(GL\_FRONT, GL\_SPECULAR, [1.0, 1.0, 1.0, 1.0])  glMaterialf(GL\_FRONT, GL\_SHININESS, 128.0)  def configure\_material\_matte():  glMaterialfv(GL\_FRONT, GL\_SPECULAR, [0.0, 0.0, 0.0, 1.0])  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, [1.0, 1.0, 0.0, 0.0])  glMaterialf(GL\_FRONT, GL\_SHININESS, 0.0)  def render\_matte\_cylinder():  glPushMatrix()  glTranslatef(-3.3, -1.5, -1.0)  configure\_material\_matte()  glColor3f(0.4, 0.4, 0.4)  glRotatef(27, 0, 0, 0)  draw\_smooth\_cylinder(0.5, 1.2, 100)  glPopMatrix()  def render\_shiny\_cone():  glPushMatrix()  glTranslatef(-1.3, -1.5, -1.0)  glRotatef(30, 1, 1, 0)  configure\_material\_shiny()  glColor3f(0.2, 0.7, 0.3)  glutSolidCone(1, 1, 20, 15);  glPopMatrix()  def render\_textured\_sphere():  glPushMatrix()  glColor4f(1, 1, 1, 0.3)  glTranslatef(1.2, -1.5, -1.0)  glRotatef(78, 1, 0, 0)  glRotatef(14, 0, 1, 0)  draw\_textured\_sphere(texture\_id, 0.6, 10, 10)  glPopMatrix()  def create\_shadow\_matrix(light\_pos, plane\_normal, point\_on\_plane):  d = np.dot(plane\_normal, light\_pos[:3]) # Берем только первые три компонента  c = np.dot(plane\_normal, point\_on\_plane)  shadow\_matrix = np.array([[d - light\_pos[0] \* plane\_normal[0], -light\_pos[0] \* plane\_normal[1], -light\_pos[0] \* plane\_normal[2], -light\_pos[0] \* c],  [-light\_pos[1] \* plane\_normal[0], d - light\_pos[1] \* plane\_normal[1], -light\_pos[1] \* plane\_normal[2], -light\_pos[1] \* c],  [-light\_pos[2] \* plane\_normal[0], -light\_pos[2] \* plane\_normal[1], d - light\_pos[2] \* plane\_normal[2], -light\_pos[2] \* c],  [-plane\_normal[0], -plane\_normal[1], -plane\_normal[2], d]])  return shadow\_matrix  def render\_shadow():  shadow\_matrix = create\_shadow\_matrix(light\_position, [0.0, 0.0, 1.0], [0.0, 0.0, -1.6])    glPushMatrix()  glMultMatrixf(shadow\_matrix.T)  # Отключаем освещение и текстуры для отрисовки теней  glDisable(GL\_LIGHTING)  glDisable(GL\_TEXTURE\_2D)    # Устанавливаем черный цвет для всех теней  glColor4f(0.0, 0.0, 0.0, 0.5)  # Прямо отрисовываем геометрию, но без материалов, чтобы они были черными  glPushMatrix()  glTranslatef(-3.3, -0.5, -1.0)  draw\_smooth\_cylinder(0.5, 1.2, 100) # Тень от цилиндра  glPopMatrix()  glPushMatrix()  glTranslatef(-1.0, -0.5, -1.0)  glRotatef(30, 1, 1, 0)  glutSolidCone(1, 1, 20, 15);  glPopMatrix()  glPushMatrix()  glTranslatef(1.5, -0.5, -1.0)  glRotatef(78, 1, 0, 0)  glRotatef(14, 0, 1, 0)  glutSolidSphere(1.0, 15, 15) # Тень от сферы  glPopMatrix()    # Включаем обратно освещение и текстуры после отрисовки теней  glEnable(GL\_LIGHTING)  glEnable(GL\_TEXTURE\_2D)    glPopMatrix()  def display():  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)  glLoadIdentity()  gluLookAt(12, 8, 30, 0, 0, 0, 0, 1, 0)  update\_light\_position()  set\_lighting\_intensity()  render\_matte\_cylinder()  render\_shiny\_cone()  render\_textured\_sphere()  render\_shadow()  draw\_plane()  glutSwapBuffers()  def update(value):  glutPostRedisplay()  glutTimerFunc(16, update, 0)  def main():  glutInit(sys.argv)  glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH)  glutInitWindowSize(1500, 500)  glutCreateWindow(b"3D Primitives with Moving Light Source and Shiny Torus")  glEnable(GL\_DEPTH\_TEST)  init\_opengl()  glutDisplayFunc(display)  glutTimerFunc(16, update, 0)  glutMainLoop()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

# Вывод

В ходе выполнения задания была успешно разработана программа для расчёта и визуализации динамических теней, отбрасываемых объектами на плоскость. Используя выбранный набор из трёх предметов, мы сохранили их свойства материалов и добавили текстурирование как объектов, так и плоскости. Внедрение одного источника освещения с возможностью изменения его положения позволило демонстрировать эффект динамических теней, создаваемых объектами в сцене. Результаты работы подтверждают эффективность предложенного подхода к визуализации освещения и теней в 3D-пространстве, что открывает возможности для дальнейшего изучения и применения в более сложных графических задачах.