**Computer Graphics Project Report: Obstacle Avoiding Game**

16075040 - Piyush Choudhary

16074008 - Khush Chopra

16074010 - Raghav Kabra

**Abstract**

The project is a game having obstacles coming towards the screen and the user have to use arrow keys to avoid and increase their score. This game is coded in Python and uses PyOpenGL libraries. The development of this project is to showcase techniques of computer graphics like texture mapping, 3D objects, translation and rotation of 3D objects, polygon filling, perspective viewing, hidden surface removal. The game is infinite and uses optimisation techniques to use minimum space as possible.

**Functionalities**

**Hidden Surface Removal**

In a 3D environment with multiple non-transparent surfaces, many surfaces can be blocked partially or completely from the perspective view of the used viewport, parts of surfaces are not visible from the user’s viewpoint. It is necessary to remove such surfaces to get the image to be presented on the computer screen. Hidden surface removal helps to decrease the processing the surfaces which are not visible on the viewport. The obstacles have surfaces which can be blocked by surfaces of other obstacles, we use hidden surface removal on all such surfaces.

**Gradient Filling**

The functionality of filling surfaces with colors, we described surfaces using the vertices making up that surface and the color associated to that vertex. Then using the PyOpenGL method of coloring surfaces was used. The library blended the colors together.

**Translation and Rotation**

The functionality of translation and rotation is achieved by forming translation and rotation matrices which are then used to form the composite matrix. Using the composite matrix, we can perform multiple types of operations in just one matrix multiplication. To move the obstacles left and right, the project uses translation. The functions of PyOpenGL have been used for this purpose.

**Perspective Projection of Objects**

Perspective projections are used to produce 2D images from 3D scenes which look natural. When we view scenes in everyday life far away items appear small relative to nearer items. The project produces a perspective view of the scene where gluPerspective specifies a viewing frustum into the world coordinate system.

**Functions Used**

**void glBegin(GLenum mode)**

glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted. Specifies the primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd.

**void glColor3f(GLfloat red, GLfloat green, GLfloat blue)**

Specify new red, green, and blue values for the current color associated with the current vertex.

**void glVertex3fv(const GLfloat \* v)**

Specify x, y, z coordinates of a new vertex which is part of a new primitive or primitives.

**void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)**

GluPerspective specifies a viewing frustum into the world coordinate system.

**void pygame.init()**

It initialize all imported pygame modules.

**void pygame.display.init()**

Initialize the display module.

**void pygame.display.set\_mode()**

Initialize a window or screen for display.

**void glTranslatef(GLfloat x, GLfloat y, GLfloat z)**

glTranslate produces a translation by x, y, z. The current matrix is multiplied by this translation matrix, with the product replacing the current matrix, as if glMultMatrix were called with the following matrix for its argument.

1 0 0 x

0 1 0 y

0 0 1 z

0 0 0 1

**void pygame.event.get()**

Pygame handles all its event messaging through an event queue. It get events from the queue

**void glGetDoublev(GLenum pname, GLdouble \* params)**

It return values for simple state variables in GL. pname is a symbolic constant indicating the state variable to be returned, and params is a pointer to an array of the indicated type in which to place the returned data.

**void glClear(GLbitfield mask)**

glClear sets the bitplane area of the window to values previously selected by glClearColor, glClearIndex, glClearDepth, glClearStencil, and glClearAccum.

**void pygame.display.flip()**

Update the full display Surface to the screen.

**void glEnd()**

glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives.

**Bibliography\**

1. <https://www.khronos.org/> PyOpenGL documentations.
2. <https://www.pygame.org/docs/> pygame documentations.
3. http://www.opengl.org/ online man pages.