Blunder: A 3D Modelling Software

Tayyab Ahmed, Sarosh Krishen

BSCS-9A, School of Electrical Engineering and Computer Science
National University of Science and Technology, Islamabad, Pakistan

tahmed.bscs19seecs@seecs.edu.pk

sarosh.bscs19seecs@seecs.edu.pk

***Abstract*— A 3D modeling software made from scratch that can create 3D geometry(objects) from points in a 3D coordinate system. Capable of rendering the objects on the screen and able to move the camera and view them from different angles. The 3D geometry can be transformed and edited interactively.**

Keywords — 3D modelling, 3D camera, Computer Generated Imagery, 3D geometry

I. Introduction

3D modeling is the process of developing a mathematical coordinate-based representation of any surface of an object (inanimate or living) in three dimensions [1]. 3D modeling is the first step in every 3D graphics application. Either it is Computer Generated Imagery (CGI), 3D games, Visual Effects in movies, animated movies, virtual reality or augmented reality, Every 3D application requires 3d Models. 3d Models provide the base for all 3d graphics in computers. These models are polished by adding materials and textures to make them look like real objects. These Models are designed in some 3d modeling software. This software allows the designer to interact with the 3d model visually instead of working with numbers and coordinates. This Project is aimed to do the same. Our aim was to create a software from scratch that allows the user to view, transform and edit 3d geometry interactively while the software handles the background calculations.

II. Language, Tools and Libraries

The programming language we used is **Python.** The reason behind this choice is the extensive mathematical libraries available with python. Some of the libraries we used in the project are

- Pygame
- Python Math Library
- Numpy Arrays
- Numpy Linear Algebra
- Python Copy

The following IDEs (Integrated Development Environment) were used, based on our personal choice:

1. MS Visual Studio Code
2. JetBrains PyCharm

III. Project Requirements and Goals

The main requirements and challenges that we kept in mind while development of this project are listed below:

- Development an Object Oriented Software
- Make a point based 3D coordinate system
- Implement a Camera to project the 3D geometry to the 2D screen
- Ability to draw the objects on the screen
- Provide a way to connect the points in 3D space using quads(4 sided polygons)
- Allow user to select quads
- Allow the user to move the selected quads
- Allow the user to rotate the selected quads
- Allow the user to scale the selected quads
- Allow user to extrude selected quads
- Project the 3d points onto the screen
- Allow multiple objects in the scene
- Allow the user to select one or more objects
- Allow the user to move selected objects
- Allow the user to rotate selected objects
- Allow the user so scale selected objects
- Get user input and pair it with camera movement and other actions
- Allowing the camera to move around the geometry to view it from different angles
- Shade the surfaces in an efficient way
- Effective way to store shading information
- Allow the user to switch between object mode and object edit mode

All of these requirements were achieved with Blunder. We provide effective solutions for all of these challenges. These solutions are discussed later in this report.

Along with functional requirements we also provided some non functional requirements. Some Non functional requirements are:

- Ability to select more than one object.
- Ability to select all objects by a shortcut.
- Ability to duplicate objects using a shortcut.
- Ability to lock the axis while moving, rotating, and scaling objects or quads, the axis for the transformation can be locked.

IV. 3D Coordinates And Linear Algebra

Linear Algebra is an obvious choice when dealing with 3D Graphics. It is very useful to represent n-dimensional points, transformations and vectors using matrices and performing operations on them. The concepts used from Linear algebra include:

- Mathematical operations with matrices
- Matrix Dot and Cross products
- Vector Space
- Linear transformations
- Projections
- Vector direction cosines
- Basis transformations
- Homogeneous Coordinates

The project uses a 3d coordinate system with the Origin (0, 0, 0) at the centre of the screen. Everything else is relative to the origin. Three basis vectors for each direction are used to draw three lines in the 3D space to represent the 3 axis. Each axis is color coded i.e

X-axis — Red
Y-axis — Green
Z-axis — Blue

V. Pygame And Rendering 2D Frames

We are using Pygame to render our 2D frames on the screen. However Pygame is in no way related to 3D geometry or 3D projection. We are doing all the 3D operations manually since that is the core of the project and using some library to do that beats the purpose of the project.

The purpose of using Pygame is to render 2D frames continuously at a fixed frame rate. We are using Pygame just to draw 2D shapes on the screen. There is a main game loop that runs infinitely while the program is running. The game loops calls 3 main functions:

A. Event Handler

Detects and handles user input from keyboard and the mouse. Handles different combination of keys and shortcuts and sets the operations to to be performed

B. Update

1. Performs the actions set by Event Handler
2. Applies the transformations to objects and quads
3. Apply camera transformations
4. Calculated projected 2D coordinates
5. Calculate shading data for each quad

C. Draw

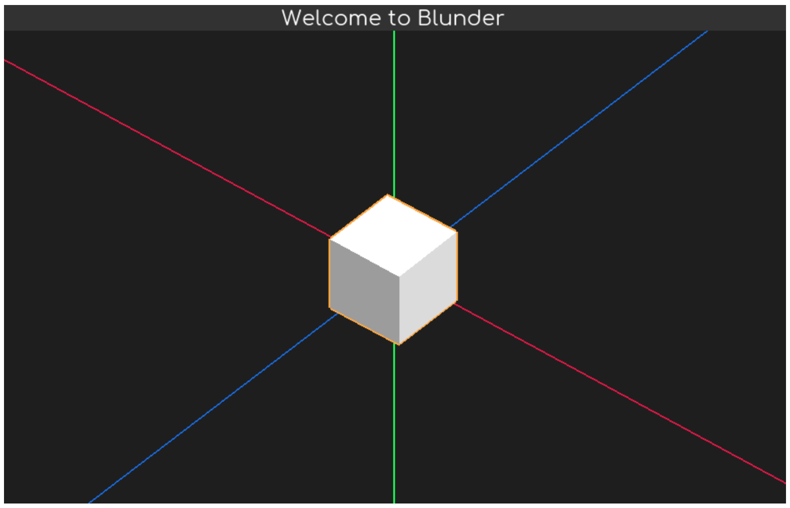
1. Puts the objects in an order to be rendered. Then orders the quads inside the object to render the closest quad to the camera on the top.
2. Draws the axes
3. Draw object selection outline
4. Draw the surfaces using the shading
5. Draw the quad selection outline

These three functions run every frame at a fixed frame rate of 60 fps. Most of the operations are optimized keeping this in mind to keep the computational load of does so many calculations 60 times a second but still there are a lot of calculations to be done that are necessary for the project to work. Beside that we are still able to achieve a fixed frame rate of 60 fps on every device we tested on.

VI. 3D Objects And Quads

We went with the object oriented approach for creating 3D objects. There is an **object_3d** class that serves as a blueprint for creating objects. This class contains all the attributes and functions related to a 3D object.

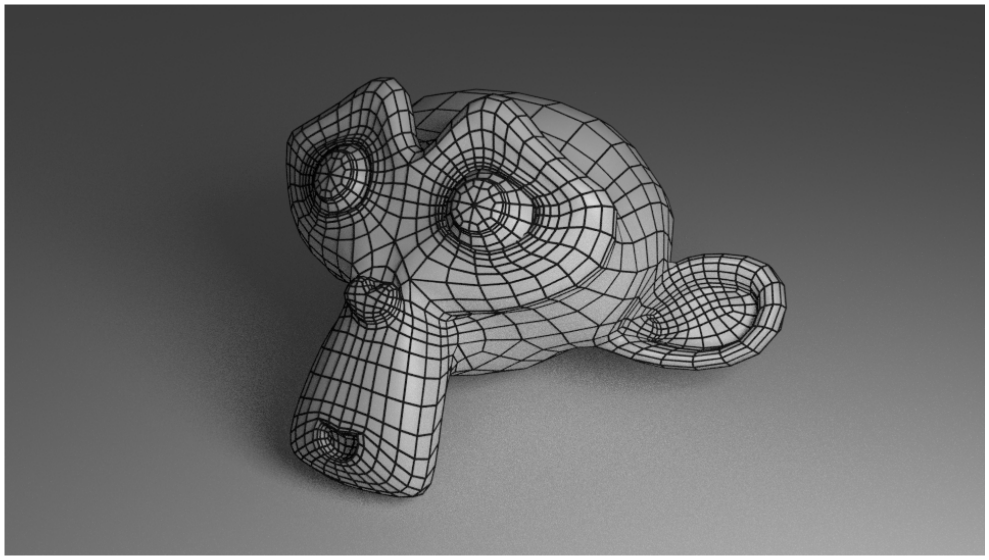
Every new object created is an instance of this class. It starts as a Cube centered at the origin with dimensions being 100 units wide, 100 units long and 100 units high. It consists of 8 points hardcoded since every object needs to start from something so a cube is the easiest to hardcode.



(a) The cube, inception of every 3D object in Blunder

The points are connected by Quads. Quads are 4 sided polygons that share points and edges to make a 3D object. The quads are stored in a Nx4 numpy array inside the object. They store the indexes of points that make them . One quad can have 4 points at most.The shape of a quad does not need to be uniform but all the points should be coplanar. The quads always share edges or else there will be no closed shape.

Once we can make quads we can use different shapes, sizes and rotations of quads to make any geometry. Most of the 3D modeling softwares use triangles or quads for the same reason since every surface can be divided into quads and quads can be subdivided into triangles. Most 3D rendering softwares use triangles since triangles are always coplanar and thus more accurately represent a surface. However 3D modeling softwares prefer quads since they are easier to edit and subdivide. We are using quads in our project for the same reason. Below (figure (b)) is an example from Blender (Blender.org) of their famous susan monkey head:



(b) Blender susan Monkey head, made from Quads

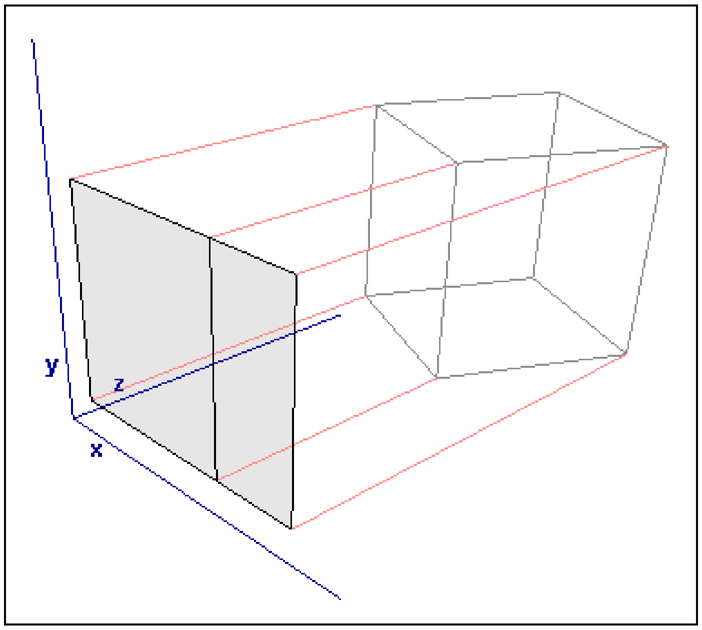
VII. Camera And 2D Projection

One of the biggest challenges is displaying 3D geometry on a 2D screen. We need to implement a Camera that provides a perspective to view the geometry. The camera should be able to produce an image from its point of view. It should also be able to move around the scene and be able to produce different perspectives from different angles.

After a lot of brainstorming and trying different approaches we came up with a solution that is simple and works pretty well. Instead of moving the camera and then calculating everything from the camera perspective, we can just keep the camera fixed and move the whole scene by using matrix transformations. This is simpler because it lets us keep the original coordinates and calculate camera coordinates at render time. The camera does not need to be in the same coordinate space. Each frame the camera transformation is applied to the whole scene and the camera coordinates are stored for each object.

The camera can be rotated by applying rotation transformation(discussed later) to everything in the scene before rendering it and it can be zoomed by applying the scale transformation(discussed later) to the whole scene. It is a bit expensive but there is no easy way around it. The user can rotate by holding and dragging the right click on the mouse and can be zoomed by using the scroll wheel.

We can use the concept of Homogeneous coordinates from Linear algebra. Homogeneous coordinates allow us to project points from higher dimension to a lower dimension. We can project our 3D coordinates into 2d by using this technique (Figure (c)). For now we fix our camera at a point in space and generate 2D coordinates at each frame. These 2D coordinates are stored in the **object_3d** object so that they could be rendered on the screen.



(c) Projection of a 3D object in 2D

These 2D coordinates are then converted to screen coordinates to adjust for the origin being in the centre of the screen. After that, first the objects are sorted from the furthest to to the closest to the camera to render them in an order and then the quads in each objects are sorted for the same purpose. At last all the quads are rendered using pygame in an order to keep the closest quads to the camera on the top.

VIII. Object And Quad Transformations

There are two modes in our software. The user can switch between these modes by using the TAB key.

A. Object Mode

In object mode, the user can select objects and apply transformations such as scale, transform, move and rotate. Multiple objects can be selected in object mode by holding the shift key and clicking on the object. In object mode all the points in an object are affected by the transformations. This can be seen as moving the entire rigid object in the scene instead of moving the points.

B. *Edit mode*

In edit mode, only one object is active and the user can select the quads of that object. Multiple quads can be selected in object mode by holding the shift key and clicking on the quads. Transformations such as scale, transform, move and rotate can be applied to the selected quads. Edit mode also allows another feature of extruding the quads. In short, edit mode can be seen as changing the geometry itself.

Object mode and Edit mode share some transformation. These transformations are applied based on what mode is active but fundamentally work the same way in both the modes. These transformations are discussed below:

A. *Translation:*

Translation is the simplest transformation. It refers to displacing the points in a certain direction. Translation can be achieved by addition. We just add the displacement along each axis to the point's coordinates

To translate $[pos_x,\ pos_y,\ pos_z]$ *by* $[disp_x,\ dis_y,\ dis_z]$:

$$new\_postion = [pos_x + disp_x,\ pos_y + dis_y,\ pos_z + dis_z]$$

B. *Rotation:*

Rotation is where things get complicated. To rotate a vector by an angle, we need to calculate a transformation matrix. We have two options here. Either we calculate separately 3 rotation matrices, one for each axis and then multiply them with the original points to get the rotation. The other way and the way we are doing it is calculating one single matrix that applies all the rotations at once. This way is computationally cheaper since matrix multiplication is more expensive than calculating sines and cosines.

$$R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma)$$

$$= \overset{\text{yaw}}{\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}} \overset{\text{pitch}}{\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}} \overset{\text{roll}}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}}$$
$$= \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}$$

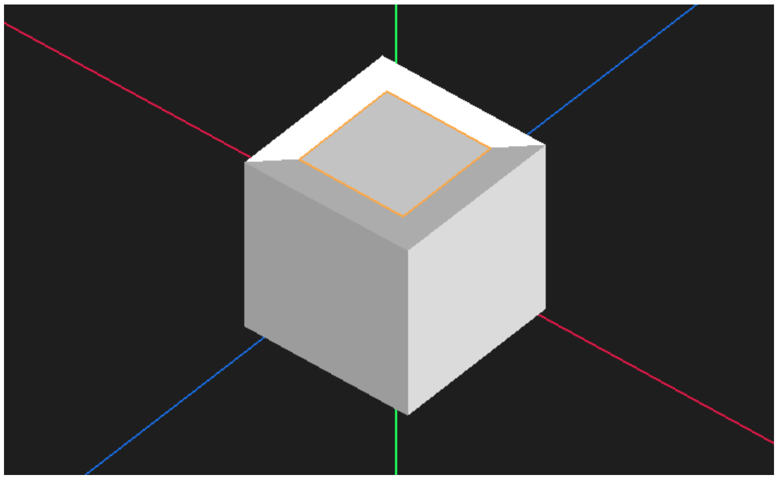
C. *Scale:*

To scale we also need a transformation matrix, we can scale along every axis by using a single transformation matrix. We can calculate this matrix by placing the scale vector in a diagonal matrix. We can multiply this matrix with the original points to transform them.

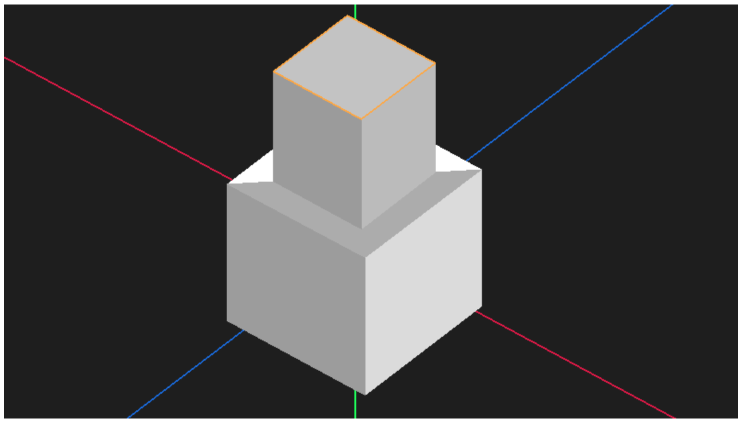
$$S_v p = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} v_x p_x \\ v_y p_y \\ v_z p_z \end{bmatrix}.$$

D. *Extrude:*

Extrude is an operation that can only be performed in edit mode. Extrude is used to duplicate a quad and connect it to the original quad, making 5 new quads. The new surface can be translated along the normal of the old surface.



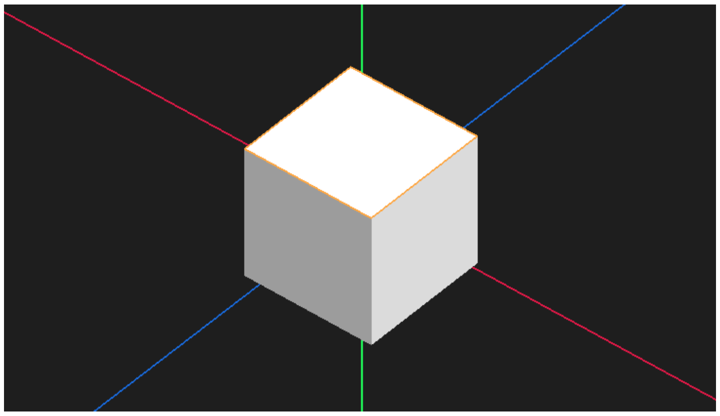
(d) Before Extruding



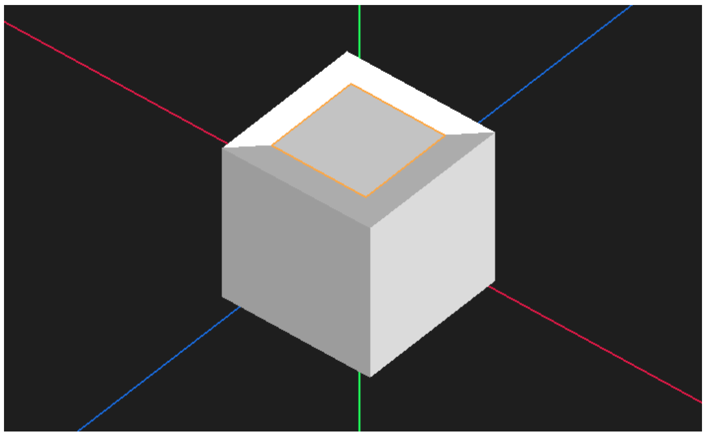
(e) After Extruding

E. Inset:

Inset also only works in edit mode. It is similar to extrude but instead of pushing the new face outward, it scales it on the surface of the original quad.



(f) before inseting



(f) after inseting

IX. Controls And Shortcuts

In Order to be able to work with Blunder, learning the controls and shortcuts is a must. The User interface is not currently in a state that we would like but it is our top priority to make it easier to use. Thus for now, following controls could be used to control the software.

1. **Select:**

Objects/Quads could be selected by left clicking on them. You can hold the shift key to select multiple objects/quads.

You can click on an empty area on the screen to deselect.

You can press A to select all of the objects in Object mode.

2. **Camera control:**

You can rotate the camera by holding the right click and dragging the mouse.

You can zoom in or out using the mouse scroll wheel.

3. **Switch Modes:**

You can press TAB to switch between Object and Edit mode.

4. **New Object:**

You can create a new object by pressing shift + N. A new object will be created at the origin.

5. **Duplicate Objects:**

You can Duplicate Objects by pressing shift + D. The object will be duplicated.

The new object will be in translation mode.

6. **Translate:**

You can translate an Object/Quads by pressing T. You can drag the mouse to control how much to move it.

You can press X, Y and Z to lock the axis.

7. **Rotate:**

You can rotate an Object/Quads by pressing R. You can drag the mouse to control how much to rotate it.

You can press X, Y and Z to lock the axis.

8. **Scale:**

You can Scale an Object/Quads by pressing S. You can drag the mouse to control how much to scale it.

You can press X, Y and Z to lock the axis.

9. **Extrude:**

You can extrude a Quads by pressing E. The new quad will be in translation mode.

10. **Inset:**

You can inset a Quad by pressing I. The new quad will be in scale mode.

X. Shortcoming And Future Improvements

We did a lot of stuff in this project but could not get to all of them here are some things that we would like to improve in the future:

- Better user interface.
- Add surface subdivision.
- Add other starter objects such as cylinders and spheres.
- Improve the shadder.
- Add textures.

References

[1] (2021) “3D Modeling - Wikipedia” (online)
Available: https://en.wikipedia.org/wiki/3D_modeling#cite_note-1