# Design documents

# Problem and Solution:

I am creating a mini 3D software for the ease of average users who want to create simple 3D objects.

## General Solution

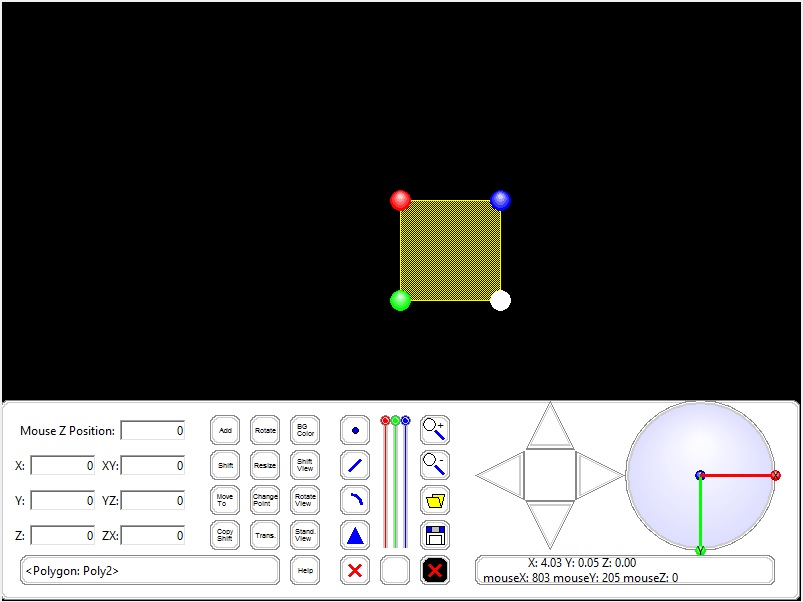
The solution involves have a 3D model involving points, lines and polygons. The points will contain the model’s coordinates and a conversion function to convert to the screen coordinates using matrix and vector calculations. The lines and polygons will contain the references to those points.

The program also allows for drawing of curved lines and curved polygons using Bezier curve function.

The mouse detection of the points is by bounding circle. The mouse detection of the lines is by the distance of the mouse to the line. The mouse detection of the polygons is depended on bounding box and ray-casting.

The whole structure of the program is split to 6 python files. The first file contain the main interface and the global variables associated with it. Then there are 3 files each corresponding to points, lines, and polygons. There is a file for main interface buttons, sliders, sphere and arrows. The last one is utility that is shared among several classes, for instance ray-casting for buttons and polygons.

## Interface



The user interface is split into several sections:

|  |  |
| --- | --- |
| Section |  |
| Screen | The objects are displayed in the screen. The objects can be created and edited here. |
| Textboxes | Contain the textboxes for users to input values and activate certain functions |
| Text-Based Buttons | The additional features on top of the normal features |
| Drawing Buttons/Sliders | Allow users to draw points, lines, curve and polygons  Allow users to change colors for objects and background |
| Menu/Control Buttons | Allow users to zoom, load and save file |
| Arrow | Allow users to change shift the view |
| Sphere | Allow users to rotate the view |
| Message boxes | Allow users to see the details of the objects selected and mouse coordinates |
| Help message | Displayed over the standard buttons |
| Tutorial | Activated by help button, provide the functions of the buttons |

# Detailed functions guide

## Utility functions:

Global functions that are used in several classes

|  |  |
| --- | --- |
| Name | Usage |
| createCircle(canvas,cx,cy,r,color,oriR,oriC=None) | Create 3D like sphere by creating smaller circles recursively and lighten the color |
| depthCmp(obj1,obj2) and depthCmp2(cood1,cood2) | Custom sort comparator to sort the list by depths depending on the type of input |
| matrixSolver(A,B) | Solve linear equations by Gaussian Elimination, used to calculate X,Y,Z from the mouse X,Y and Z. |
| dotProduct(A,B) | Calculate the dot product of 2 vectors |
| rotate(x,y,theta) | Return the new coordinates after rotation by angle theta |
| rgbString(red, green, blue) | Return the string representation of color using integers |
| beizerCurve(x0,y0,x1,y1,x2,y2) | Get the list of fine coordinates for beizer curve using 3 points and taking the center one as the control point. |
| hitTestLine(point,mouse,r) | Given a list of coordinates (point), check if the mouse is in the line |
| hitTestCircle(x0,y0,r,mouse) | Give a circle with center x0, y0 and radius r, check if the mouse is in the circle |
| boundBox(listC,mouse) | Create the bounding box using coordinates and check if the mouse is in the box |
| rayCast(point,mouse) | Use Ray-casting algorithm (odd-even rule) to check if the mouse is in a polygon |

## Class Point:

The class for the point object in the 3D mini-modelling

|  |  |
| --- | --- |
| Name | Usage |
| getPointList(cls) | Get the list of points currently in the point class |
| getPointCount(cls) | Get the max counter for the points |
| \_\_init\_\_(self,x,y,z,name,color,mainInt) | Initiate the point |
| modelToXYConversion(self) | Convert its X,Y,Z in the model to screen X,Y and Z |
| draw(self) | Draw the point |
| hitTest(self) | Detect for mouse over for the point |

## Class tempPoint:

The class for the temporary point (mouse point) only during drawing lines and polygons so that it previews the drawing of the line/polygon

|  |  |
| --- | --- |
| Name | Usage |
| \_\_init\_\_(self,x,y,z,name,color,mainInt) | Initiate the point |
| modelToXYConversion(self) | Convert its X,Y,Z in the model to screen X,Y and Z |
| draw(self) | Draw the point |

## Class Line:

The class for line/curve contain references to points.

|  |  |
| --- | --- |
| Name | Usage |
| getLineList(cls) | Get all the line references in a list |
| getLineCount(cls) | Get the counter for the line |
| \_\_init\_\_(self,listP,name,color,mainInt,width=5) | Initiate the line  All lines have 2 points  All curves have 3 points with the center as the control points |
| step(self) | Compute the list of screen coordinates from the list of points   * Straight line: 2 points * Curve: Use Bezier curve function   Compute depth of the line |
| draw(self) | Draw the line |
| hitTest(self) | Detect collision of the line using the utility functions |

## Class tempLine:

The class for a temporary line that is visible as a preview for drawing the line/curve

|  |  |
| --- | --- |
| Name | Usage |
| \_\_init\_\_(self,listP,name,color,mainInt,width=1) | Initiate the line |
| step(self) | Compute the list of coordinates from the list of points |
| draw(self) | Draw the temporary line |

## Class Polygon:

The class for polygons and curved polygons which contains references to points

|  |  |
| --- | --- |
| Name | Usage |
| getPolyList(cls) | Get the list of polygon references for all polygons |
| getPolyCount(cls) | Get the counter for the polygons |
| \_\_init\_\_(self,listP,name,color,mainInt,width=5) | Initiate the polygon |
| step(self) | Compute the list of screen coordinates from the list of points   * Straight line part of polygon * Curved part of polygon : Use Bezier curve function   Compute depth of the polygons |
| draw(self) | Draw the polygon using the screen coordinates |
| hitTest(self) | Detect mouse over event for the polygon using raycast |

## Class tempPolygon:

The class for temporary polygons which help user preview drawing of polygons

|  |  |
| --- | --- |
| Name | Usage |
| \_\_init\_\_(self,listP,name,color,mainInt,width=3) | Initiate the polygon |
| step(self) | Compute the list of screen coordinates from the list of points   * Straight line part of polygon * Curved part of polygon : Use Bezier curve function |
| draw(self) | Draw the polygon using the screen coordinates |

## Class MyButton:

The class for buttons with the white/green combo

|  |  |
| --- | --- |
| Name | Usage |
| getButtonList(cls) | Get the list of button references |
| \_\_init\_\_(self,w,h,x0,y0,main,name="") | Initiate the button |
| draw(self) | Draw the button |
| drawIndButtons(self,x0,y0,x1,y1) | Individual icon/text drawing for the button |
| roundedButtonCood(self,x0,y0,x1,y1) | Compute the round button coordinates in a polygon |
| buttonPressed(self) | Mouse event for button |
| hitTest(self) | Mouse over detection for button |

## Class MyTutorial (MyButton):

The class for tutorial

|  |  |
| --- | --- |
| Name | Usage |
| getTutorialList(cls) | Get the list for tutorial references |
| \_\_init\_\_(self,w,h,x0,y0,main,name="") | Initiate the tutorial |
| draw(self) | Draw the tutorial box and text |
| drawArrow(self,buttonName) | Draw the red arrow to point to the buttons |
| drawBox(self) | Draw the red bounding box showing the textboxes |

## Class MyArrow (MyButton):

The class for arrows to move around in 3D view

|  |  |
| --- | --- |
| Name | Usage |
| getArrowList(cls) | Get the list for arrow references |
| \_\_init\_\_(self,w,h,x0,y0,direction,main) | Initiate the arrows |
| draw(self) | Draw the arrows |
| roundedArrowCood(self,x0,y0,x1,y1) | Compute the arrow coordinates on screen |

## Class Sphere:

The class for the round x,y,z-axis panel

|  |  |
| --- | --- |
| Name | Usage |
| getSphereList(cls) | Get the list of references for sphere |
| \_\_init\_\_(self,r,x0,y0,main) | Initiate the sphere |
| draw(self) | Draw the sphere |
| hitTest(self) | Detect mouse over event for the sphere |

## Class MySlider:

The class for the color sliders

|  |  |
| --- | --- |
| Name | Usage |
| getSliderList(cls) | Get the list of references for the slider |
| \_\_init\_\_(self,r,x,y,size,x0,y0,main,color) | Initiate the color slider |
| draw(self) | Draw the sliders |
| hitTest(self) | Detect mouse over for the slider buttons |

## Class MainInterface:

|  |  |
| --- | --- |
| Name | Usage |
| \_\_init\_\_(self,width,height) | Initiate the main interface |
| run(self, width=800, height=600) | Run the main interface |
| textBoxCreation(self,root) | Create the 7 textboxes |
| Initiation Functions   * init(self) * initMouseVar(self) * initHelpMsg(self) * initLists(self) * initKey(self) * modeReset(self) * initColor(self) * initLoadSave(self) * initInterface(self) * myOtherInterfaceCreation(self) * myButtonCreation(self) * demoCube(self) * tempObjects(self) | Various initiation functions for mouse, key, help, load, save, interface and other important variables. |
| Regular time interval functions   * timerFired(self) * arrowShift(self) * sphereRotate(self) * XYtoModel(self,x=None,y=None,z=None) * dragging(self) * keyPressTime(self) | These events is used to handle events such as key/mouse hold down as well as dragging. |
| Mouse Events   * mousePressed(self,event) * objectMousePressed(self) * mouseReleased(self,event) | Various mouse events for the interface, objects and the mouse release event. |
| Key Events   * keyPressed(self, event) * keyPressedTutorial(self,event) * keyReleased(self, event) | Various key press events, including a special function for key press tutorial. |
| Draw functions:   * redrawAll(self) * drawObjects(self) * drawInterface(self) * drawHelpMessage(self) * drawTextBoxLabels(self) * drawAxis(self) | The drawing on the canvas draw 2 important parts, the objects and the interface.  For the objects, the draw function of point, line and polygon is called.  For the interface, the various buttons, sliders, arrows and sphere have to be drawn. Furthermore, the help message and the textbox labels must be drawn as well. |
| Object creation   * addPoint(self) * addPointByText(self) * addLine(self) * addCurve(self) * getNewTempPoint(self) * addLineObject(self,lineType) * addPolygon(self,enterPressed=False) | Add point is quite simple. The 2 types are normal add on the screen and via textbox inputs.  Add line and polygon use a temporary storage to store the point references. If the user click an empty spot, a new point will be created. |
| Object modification   * shiftPointByText(self) * movePointByText(self) * copyPointByText(self) * rotatePointByText(self) * rotatePoints(self,p,xy,yz,zx,cx,cy,cz) * resizePointByText(self) * changeColor(self) * removeInstance(self,remLast=None) | This part is about object modification by textboxes.  Shift: Shift the object by a certain amount  Move: Move the center the object to a certain point  Copy and Shift: Copy the object and shift it  Rotate: Rotate the object by its center  Resize: Resize the object by its center  Change color: Change the object’s color  Remove instance: Delete the object by removing its references to the class and other objects  \*Note: Rotate and Resize a point gives back the same point because the center never change and the point is of size 0. |
| Global modification   * standardView(self) * shiftView(self) * rotateView(self) * bgColorChange(self) * changePointSize(self) * changeTransparency(self) * zoom(self,zoomAmount) | There are view modifications such as shifting and rotating.  There are also other global changes such as point size, transparency of polygons, zoom and background color. |
| Load and Save   * deleteObjects(self) * loadFile(self) * loadFromString(self, newLoad) * loadPoints(self,newSections) * loadLines(self,newSections) * loadPoly(self,newSections) * loadRest(self,newSections) * saveFile(self) * saveString(self) * saveObjects(self,newSave) * saveGlobals(self,newSave) | The objects and the global variables are saved in a string and hashed to prevent corruption.  When the file is loaded, the string is split into lists to get the original values back. |
| Tutorial   * tutorialInit(self) * tutorialEnd(self) | Initiate the tutorial box and end it. |
| Others:   * pointOutOfBound(self,x,y,z) * viewOutOfBound(self,x,y,z) * determineColor(self) * rotateXY(self,theta) * rotateYZ(self,theta) * rotateZX(self,theta) * shift(self,x,y,z) * checkForNumber(self) | There are boundaries check to ensure that there will not be stack overflow like the bug that Wang Zhiyu caused.  The rotation functions and shift functions are view modifier.  The check for number automatically prevent users from entering nonsensical values. |

## Citations:

1. Animation: http://www.kosbie.net/cmu/fall-13/15-112/handouts/Animation.py
2. Non-resizeble: <http://www.kosbie.net/cmu/fall-11/15-112/handouts/misc-demos/src/resizableDemo.py>
3. Mouse Motion: <http://www.kosbie.net/cmu/fall-11/15-112/handouts/misc-demos/src/mouseMotionEventsDemo.py>
4. Mouse Event: <http://www.kosbie.net/cmu/fall-11/15-112/handouts/misc-demos/src/mouseEventsDemo.py>
5. Key Event: <http://www.kosbie.net/cmu/fall-11/15-112/handouts/misc-demos/src/keyEventsDemo.py>
6. Entry textbox: <http://effbot.org/tkinterbook/entry.htm>
7. Load and Save: <http://tkinter.unpythonic.net/wiki/tkFileDialog>
8. Pickle: <http://docs.python.org/2/library/pickle.html>
9. Stipple: <http://www.kosbie.net/cmu/fall-11/15-112/handouts/misc-demos/src/semi-transparent-stipple-demo.py>
10. “rgbString”: **Copied** from http://www.cs.cmu.edu/~112
11. Bezier curve equation: <http://stackoverflow.com/questions/6711707/draw-a-quadratic-bezier-curve-through-three-given-points>
12. Ray-casting: <http://rosettacode.org/wiki/Ray-casting_algorithm>
13. Scalar projection: Essential Calculus. Early Transcendentals.

## Thanks:

Joel Choo: For being an awesome mentor

Wang Zhiyu: For finding bugs in my programs

Other CAs: For testing out programs

Everyone else: For testing out programs