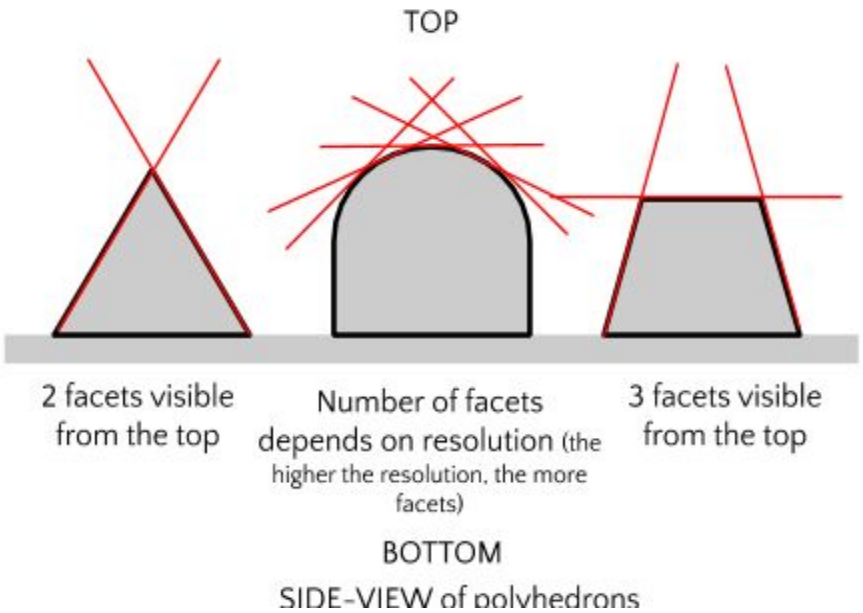
Design Document: *Applicolator*

Description of the program

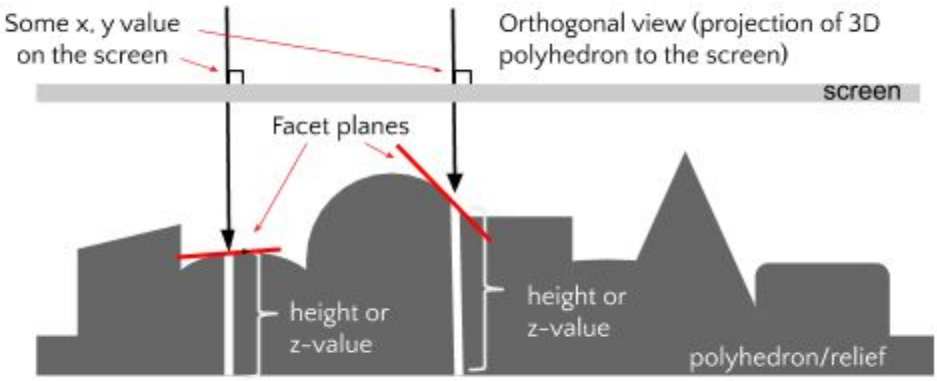
Applicolator displays a 3-dimensional polygon mesh. When the user inputs *x,y*-values with their cursor position, Applicolator highlights the visible facet that the cursor is currently hovering over and displays its normal vector with a magnitude proportional to the z-value. In order to efficiently locate the visible facet, Applicolator first checks if the x,y-point is in the convex hull of the polyhedron's projection on the x,y-plane. Second, it builds an interval tree of the smallest enclosing rectangles of facets such that the k facets whose enclosing rectangle the input x,y-values is in can be located in O(k log n) time. Third, for these rectangles, Applicolator uses the ray-casting algorithm to test whether the x,y-values are in the facet. Fourth, Applicolator compares all facets that the x,y-values belong to for the largest z-value. The applicate (z-value) is calculated for the *x,y*-point by using the normal vector and a point on the facet. The normal vector is calculated by taking the cross product of two vectors obtained from three points on the facet. In this way, the facet that is "on top" can be identified because it will have the greatest z-value. The name *Applicolator* is derived from *The Applicate (z-value) Interpolator for a Facet of a 3-Dimensional Polygon Mesh Given the Abscissa and Ordinate*.



In computer graphics, one way to define a 3D shape is through a shell/boundary model, which represents the surfaces of the 3D shape like an infinitesimally thin eggshell. Polygon meshes are commonly used, which represents the surfaces of the 3D shape using polygons.

The polygons, also called facets, may be triangles, quadrilaterals, or simple convex polygons. A polygon mesh can be stored in different file formats; Applicolator will be importing .obj files.

A 3D polygon mesh model describes its facets using its own coordinate system, called the object coordinate system. Different .obj files use coordinate systems that represent the length in different units, and the coordinate origin may be far away from the object's center. Applicolator scales and translates the object coordinate system to make the shape fits properly on the computer screen.



Side-view of screen and polyhedron/relief

A polygon mesh will be displayed in an orthogonal view so that x,y-values on the screen match with a scaled version of the polygon mesh. The corresponding applicate (z-value) to the x,y-value can be determined using vector calculations. Applicolator will assume that the z-value is zero at the vertex of the polyhedron containing the smallest z-value and normalize the magnitude of the displayed normal vector to be between zero and one. The ideal polygon mesh for this program is a 3D relief, as reliefs only have one side with 3D detailing.

Essentially, Applicolator is meant to be an interpolator for the z-value given an *x,y*-value on a plane. In order to achieve this purpose, intermediate calculations are required. These intermediate calculations have, in and of themselves, become useful features.

Features

Program:

- Identifies and highlights the visible facet that the user's cursor is hovering over
- Interpolates the z-values, given an *x,y*-value on the facet
- Display the normal vector of a facet. The magnitude of the vector is first normalized, and then set to the z-value at the *x,y*-point
- When the user checks off a checkbox, they are able to colour the facets. They can change their colour by turning the colour knobs. In this way, they can create a pretty, coloured 3D shape.

GUI:

- Contain different choices for polyhedrons and reliefs to select from
- Contain different background textures to select from
- Contain an option to check off that allows the user to "colour" the facets for fun
- Contain an RGB colour wheel to allow the user to change the colour of the polyhedron and facets

Inputs to the program

- An *x,y*-value that is the cursor's position on the screen
- GUI inputs
    - Checkbox
        - 3D colouring exercise
        - If checked, stop redrawing polyhedron and background
    - Dropdowns
        - Texture
        - Polyhedron
    - Knobs
        - Change the RGB values for the facet and polyhedron

Sketch of the User Interface

Slider to change features such as the colour, relief height, length of the pegs being shown

x3

Texture option (image button)	Texture option (image button)	Texture option (image button)	Texture option (image button)
Pshape option (image button)	Pshape option (image button)	Pshape option (image button)	Pshape option (image button)
Pshape option (image button)	Pshape option (image button)	Pshape option (image button)	Pshape option (image button)

Coding structure

ShapeGeometry2 Class

General Overview:

Responsible for:

1) **Preprocessing the .obj file**
    - Recursively find all facets, recursively find all vertices
    - Finding min/max z-values

2) **Storing facets and their smallest enclosing rectangle**
   - Create an interval tree for the facet's smallest enclosing rectangle (the top-left-most, top-right-most, bottom-left-most, bottom-right-most points)
     - Given an x,y-value, find all the rectangles that it overlaps.

Rules for Interval tree algorithm:

i) The tree is ordered by leftmost x's value.
ii) Each node has a maxSubtreeX value which dictates the largest value of x in the subtree rooted at that particular node.
iii) If your point x overlaps with the current node's shape's interval, add the interval to the result.
iv) Recur the left child.
v) Recur for the right child if your x value is larger than the right child's low.

3) **Find visible facets**
   1) Calls convex hull, tests if a point is in the polyhedron's projection onto the x,y-plane
   2) Search the interval tree for all the rectangles that enclose the point
   3) Test if the point is within the facet using the ray-casting algorithm
   4) Calculate the z-value for each point on the facet
   5) Compare z-values, the point with the largest z-value is the facet that is "on top"

4) **Handles vector calculations**
   - Calculates the normal vector
   - Calculate the z-value for an x,y-point

Fields	
PShape convexHull;	Creates shape using points from the convex hull.
float minX, minY, maxX, maxY, maxZ, minZ;	Finds the extreme values from points in convex hull (fewer points to check).
XIntervalTree facetIntervalTree;	Creates an object.

Methods	
ShapeGeometry2()	Empty constructor.
ShapeGeometry2(PShape shape)	Calls the other methods. Initializes fields for the given shape.
void getAllVerticesFromShape(PShape a, HashSet<PVector> v)	Recursive function. Adds all the shape's vertices to a HashSet to ensure no duplicate values.
private void getAllFacetsFromShape(PShape shape, ArrayList<PShape> allFacets)	Recursive function. Checks if the shape is a facet. If the shape is a facet (the family will have the same constant field value as GEOMETRY), add the facet to the ArrayList. Otherwise, do a recursive call for each child of the shape.

void initializeMinMaxXYZ(HashSet<PVector> vertices)	Initializes field values.
void initializeConvexHull(HashSet <PVector> allVertices)	Creates convex hull from an ArrayList of the polygon's vertices. Initializes the field value.
float getMinX/getMaxX/getMinY/getMaxY/getMinZ/getMaxZ() / PShape getConvexHull()	Multiple methods that are of the same type. Returns the specified value. Its purpose is to be accessed from other classes.
PShape getVisibleFacet(PVector xy)	First, check if the cursor is within the smallest enclosing rectangle formed by the shape's four most extreme points. Next, check if the cursor is within the convex hull. Return the final facet with highest z-value or null if the cursor is not on any facet.
boolean isInFacet(PShape shape, PVector xy)	Checks if a point is in a polygon using a ray-casting algorithm https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/
void setZForXYonPlane(final PVector normal, final PVector position, PVector xy)	Uses vector calculations to find the z-value given the normal vector, position vector and an xy value.
PVector getNormalVector(final PVector a, final PVector b, final PVector c)	Given 3 points on the plane, get the plane's normal vector.
PVector getNormalVectorAndSetHeight(PShape facet, PVector xy)	Set z-value for cursor position on the screen. Return normal vector. Calls setZForXYonPlane and getNormalVector.

MotorController Class

General Overview:

For the future continuation of this project:

- With the eventual goal of making a tactile mouse so that blind people can feel shape geometries of digital objects
- Get the heights for the motors, drives the motors
- Currently, this class does not contribute to the program

Fields	
PVector p1, p2, p3;	Positions of each motor relative to the cursor.
PVector mouseXY;	Where the cursor is.

Methods	
MotorController(PVector p1, PVector p2, PVector p3)	Constructor.
void moveMotor(float h, PVector normal)	Calls setZForXYonPlane.
void setZForXYonPlane(final PVector normal, PVector xy)	Sets z-values for a motor.
float getMotorHeight(int motor)	Return motor height.

DirtyWork Class

General Overview:

- Draw relief, highlight the facet, and draw the normal vector that uses object coordinates on the screen
- Convert the mouseXY position from the screen coordinates to object coordinate
- When the cursor changes position, the process above is repeated and Applicolator then redraws the GUI

Fields	
PShape shape;	Relief or polyhedron.
ShapeGeometry2 sg;	Geometry data of the relief.
float scaleFactor;	Object coordinate to screen coordinate scale multiplier.
MotorController motorController;	Motor data. Not in use in the program
PVector mouseXY;	Current cursor position.
Float normalVectorMagnitudeFactor;	Normal vector magnitude multiplier
float motorHeightFactor	Not in use in the program.

Methods	
void printStates()	Troubleshooting.
void initialize(PShape shape)	Initializes the field scaleFactor. The polyhedron will be scaled on the screen appropriately. Initializes shape geometry.
void mouseMoves()	Call other methods depending on the cursor position.
void drawRelief()	Colours and draws the polyhedron
void highlightFacet(PShape facet)	Colour the facet the cursor is on.
void drawNormalVector(PVector normal, PVector mouseXY)	Draws the normal vector for the facet.
void drawWithObjectCoordinate(PShape shape)	Draw the shape that uses object coordinates on the screen.
void screenToObjectCoordinate(PVector xy)	Change screen space to object space for the cursor.

Convex Hull Class

General overview:

Implement the Graham Scan algorithm to find the convex hull of a set of vertices.

Fields (none)

Methods	
ConvexHull()	Empty constructor.
ArrayList<PVector> getConvexHull(HashSet<PVector> v)	Return the convex hull of the provided set of vertices, which must be distinct in terms of x and y.
int indexOfBottomestPoint(ArrayList<PVector> points)	Returns the index of the point with the smallest 'y' value
swap(ArrayList<PVector> points, int i, int j)	Swap the points at the two indexes in the array
float counterClockWise (PVector o, PVector a, PVector b)	Position vectors 'a' and 'b' Calculate the cross product of vectors 'a' and 'b' If ccw > 0, then counterclockwise, or < 180 degrees If ccw < 0, then clockwise, or > 180 degrees If ccw = 0, then collinear
float calculateDistance(PVector a, PVector b)	Return the distance between the two points.
ArrayList<PVector> grahamScan(ArrayList<PVector> points)	Graham Scan algorithm: https://en.wikipedia.org/wiki/Graham_scan

	Take in an array of points sorted by polar angles The point at index 0 is the origin Returns the convex hull, the points are in the same order as in the input array
ArrayList<PVector> sortByPolarAngle(ArrayList<PVector> points)	Take the ArrayList 'points' and sort them based on: 1. The point at index 0 is the origin 2. Find position vectors for all points using the newly established origin 3. Sort points based on their vectors' polar angles
ArrayList<PVector> mergeSort(PVector o, ArrayList<PVector> v, int start, int end)	Sorts the vectors in the specified range by polar angle. The vectors are from the point 'o' to the points in the ArrayList v. Both start and end are inclusive.
ArrayList<PVector> merge(PVector o, ArrayList<PVector> a, ArrayList<PVector> b)	Merge the two arrayLists.

XIntervalTree Class

General overview:

Interval tree of rectangles. The key is the x1 of the rectangles. The interval refers to [x1, x2]. A node may contain multiple shapes with the same x1; these shapes are stored in the list swer. The k number of smallest-enclosing rectangles that a point (x,y-value from the cursor) is within can be found in O(k log n) time where n is the number of nodes.

Fields	
XIntervalTreeNode root;	Root of the interval tree.
int numNodes, numRectangles;	The number of nodes in tree and number of enclosing rectangles. For debugging purposes

Methods	
ArrayList<PVector> getConvexHull(HashSet<PVector> v)	Return the convex hull of the provided set of vertices, which must be distinct in terms of x and y.
XIntervalTreeNode buildTree(ArrayList<ShapeWithEnclosingRectangle> swer, int begin, int end)	Build the interval tree using the elements of swer in the range of begin to end. Begin is inclusive, end is exclusive. Swer has been sorted by x1. Recursively build the tree. Get the middle node every level. If you left and right and the values for x1 are the same, add it to the current node. Recursively build left and right side of the tree. Update maxsSubtreeX.
ArrayList<ShapeWithEnclosingRectangle> getAllRectanglesOverlapXY(float x, float y)	Return all rectangles that overlap with the provided x and y values.
void getAllRectanglesOverlapXY(XIntervalTreeNode node, float x, float y, ArrayList<ShapeWithEnclosingRectangle> result)	Add rectangles that overlap with the provided x and y to results. Recursively gets called for each side of the tree.
void printTree(boolean verbose)	For testing purposes.
void printTree(String indent, XIntervalTreeNode root, boolean verbose)	

ShapeWithEnclosingRectangle Class

Fields	
PShape shape;	Facet in question.
float x1, x2, y1, y2;	Coordinates of the smallest enclosing rectangle

Methods	
ShapeWithEnclosingRectangle(PShape shape)	Constructor.
boolean overlapX(float x)	Whether the given x is within x-range of the smallest enclosing rectangle
boolean overlapY(float y)	Whether the given y is within y-range of the smallest enclosing rectangle
public int compareTo(ShapeWithEnclosingRectangle other)	Used by Collections.sort to sort objects of this class by x1
void printStates()	For troubleshooting.

XIntervalTreeNode Class

General Overview:

A node on the interval tree. The key is the x1 of the rectangle. The interval refers to [x1, x2]. A node may contain multiple shapes with the same x1; these shapes are stored in the list swer.

Fields	
XIntervalTreeNode left, right;	Left and right child node of the current node.
ArrayList<ShapeWithEnclosingRectangle> swer;	All enclosing rectangles with the same x1.
float maxSubtreeX;	Max value of x2 in all nodes under the current node.
float x1;	The leftmost x-value, same for all enclosing rectangles

Methods	
XIntervalTreeNode(ShapeWithEnclosingRectangle r)	Constructor.
void add(ShapeWithEnclosingRectangle r)	Add the rectangle of the same x1 to the node.
void printStates()	Troubleshooting purposes.

Restrictions and Requirements

The requirement for the ideal 3D polyhedron for Applicolator:

- From a top view, all facets of interest must be visible
- .obj files for polygon meshes cannot exceed the size that Processing allows with minimal lagging
    - If file sizes are too large, reduce file size by reducing the number of facets in the polygon mesh using a tool such as Blender