The Model

- ***Overview****: The model is a 1D array of vertices where each vertex represents a corner of the target object. Its main objective is keeping target object within by utilizing the Information presented in Histogram of errors given as an input by another module.*

- ***Parameters****: There are a set of parameters that controls and stabilizes the model that depends on the camera used.*
    - ***degree_threshold****: If difference in angles between 2 connected edges Is less than it, the intersecting point is removed.*
    - ***force_threshold****: Determines whether the model needs to deform or not.*
    - ***length_threshold:*** *If the length of an edge is less than it, a point in that edge is removed.*
    - ***limit****: Determines whether to insert a vertex in the middle of an edge.*
    - ***max_model_size****: Max number of vertices the model can hold.*
    - ***max_forces_size****: Max size of the array of forces (twice max_model_size)*

- ***The model class:***

    - ***Data****:*
        1. ***(point * vec ;)*** *An array of points with constant size.*
        2. ***(Float *force_ve ;)*** *An array of floating point numbers for left and right forces generated on each edge (twice the size of the points array).*
        3. ***(Float *temp_vec ;)*** *An array of floating point numbers for holding normal angle with the horizontal of each edge.*
        4. ***(point current_centroid ;)*** *a point that keeps track of current centroid.*
        5. ***(Uint32 elements_size ;)*** *an integer that keeps track of number points within the model.*

    - ***Vector functions:***
        1. ***model(void);*** *Empty constructor that allocates the required memory for the model.*
        2. ***~model(void);*** *A simple destructor that deallocates used memory in the heap.*
        3. ***UINT32 get_size(void)const;*** *returns number of points within the model.*
        4. ***point& operator[](const UINT32) const;*** *Constant accessor.*
        5. ***point& operator[](const UINT32);*** *Variable accessor.*
        6. ***BOOL insert(const point&, UINT32);*** *Inserts a point at an index and updates elements_size, If elements_size is equal to max_model_size the point isn't inserted.*
        7. ***BOOL remove(UINT32 wanted_index);*** *removes a point at an index and updates elemnts_size.*
        8. ***void reinit(const point pt);*** *This function reinitializes the model as a square of length 40, its center is the input point, if input point is (-1,-1) it reinitializes with center equal to the previous centroid.*

- ***Mathematical functions:***

1. ***BOOL forces(INT32* histogram);***
    a. *This function calculates left and right forces generated on each edge and fills force_vec with the results such that the left and right forces of the ith edge are accessed in the following way consecutively: force_vec[2*i+0], force_vec[2*i+1].*
    b. *Parameters: histogram of errors.*

2. ***BOOL deform(INT32* histogram);***
    a. *This function deforms the model to fit the object based on forces stored in force_vec within the function temp_vec is filled with with angle of the normal of each edge.*
    b. *Parameters: Histogram of errors.*

3. ***void insert_vertecies_at_position(INT32* histogram);***
    a. *This function inserts a vertex in the middle of a stable edge if error in any of its regions is greater than limit.*
    b. *Parameters: histogram of errors.*

4. ***BOOL remove_verticies_in_middle(void);*** *This function removes the vertex connecting 2 consecutive edges if their angles with the normal are "equal".*

5. ***BOOL remove_spikes(void);*** *This function removes vertices that cause the model to have spikes (very small angle between 2 edges)*

6. ***BOOL remove_small_lengths(void);*** *This function removes vertices lying on edges with minimal lengths.*

7. ***INT32 get_state(INT32* hist);***
    a. *This function is a control system that controls the model next action in the frame based on information presented in histogram of errors.*
    b. *It determines whether to deform, insert or if there was a fault (I.e., Area of the model is less than a certain number of pixels) and the model need to reinitialize.*
    c. *It updates current_centroid.*
    d. *It returns set of enum values representing the function to be called or if it's stable and no need for more operations*
    e. *Parameters: Histogram of errors*

8. ***FLOAT area(void)const;*** *This function returns area of the model.*

9. ***point centroid(void)const;*** *This function returns centroid of the model.*

10. ***BOOL intersection_check(const point& p0, const point& p1, const point& p2, const point& p3, point& inter_p) const;***
    a. *This function checks whether 2 edges represented by [p0,p1,p2,p3] are intersected in a point and the intersection lies within the edges (i.e, they are actually intersected not that the infinite lines formed by them are intersected)*
    b. *If so it filles inter_p with the intersection point of the 2 edges.*

11. ***BOOL fix_intersection(void);***
    a. *This function removes intersections between each 2 edges (they don't have to be consecutive).*
    b. *From testing the behavior of model, intersections are formed due to noise in the image which causes 2 close vertices to deform in the wrong way hence causes intersection, so this function removes end vertex of the first edge and starting vertex of second edge is replaced with the intersection point.*
    c. *If an intersection was fixed it returns true.*

12. ***void move(const point& displacement);***
    a. *This function moves vertices of the model with the displacement caused by moving the camera since when the camera moves the coordinates moves and the model stays at previous location in old coordinates.*
    b. *Parameters: point representing displacement that the camera moved in pixels.*