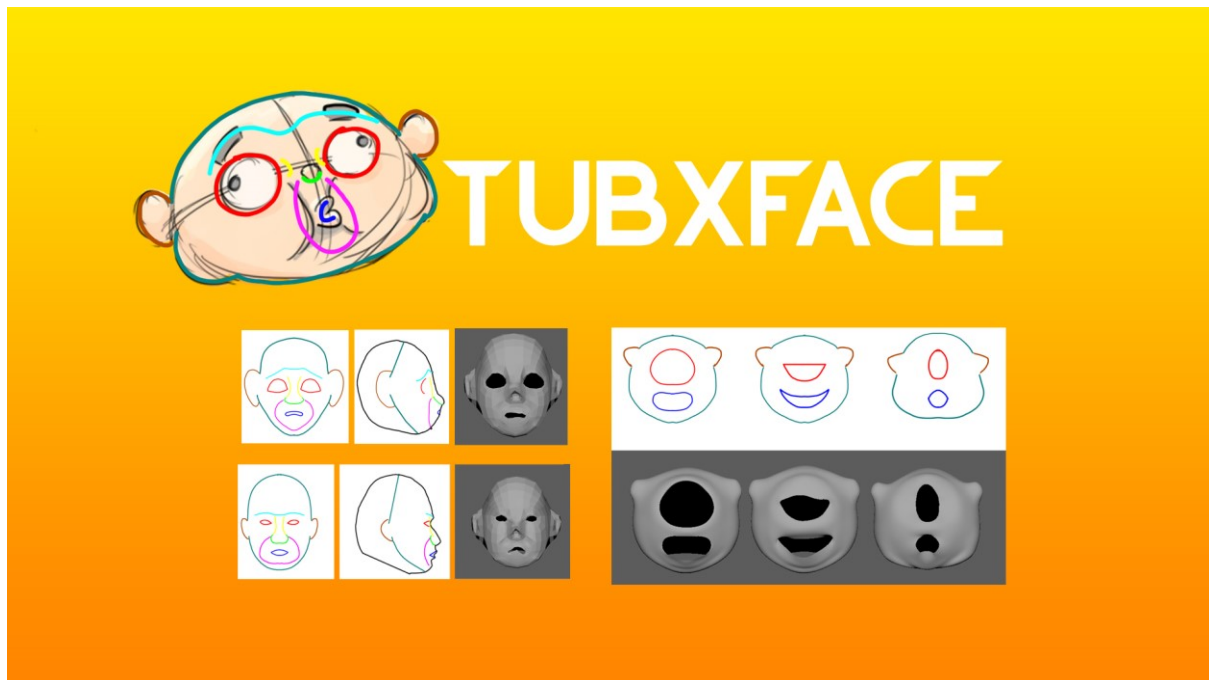


# Exploring the use of 2D orthographic images to manipulate 3D CG heads



## Abstract

The process of 3D modelling often requires a base model first where it is then refined to the finish product. This process of creating the base model often involves pushing and pulling vertices directly in 3D space.

In my innovations, I explore other ways of manipulating 3D geometries apart from the WIMP (Window, Icon, Menu, Pointer) paradigm. I then create an experimental tool to translate a front and side orthographic drawing, together with a selected 3D library geometry, to generate a basic head mesh for the artist as a starting point to continue refining.

## Introduction

When one talks about modelling or manipulating 3D geometry, they are inclined to think of the process of moving vertices directly in 3D space. Some examples of these are box (Fig1), plane or NURBS modelling which follows the WIMP (Window, Icon, Menu, Pointer) paradigm (Olsen, Samavati, Sousa, Jorge, 2009, p.85). This process, though powerful and allows for lots of control, may be daunting to a novice as one would require considerable amount of effort and expertise.

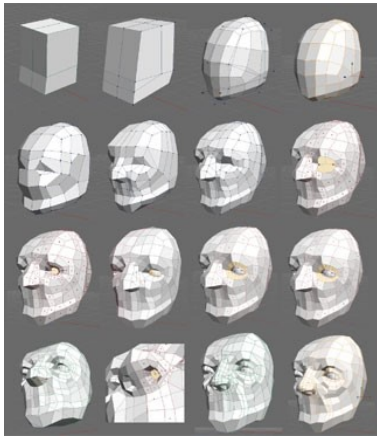


Fig1 - Box Modelling is a type of modelling where the artist starts off with a primitive and slowly refines it till the end product is achieved. (Picture by Clark 2014)

There are many other ways, however, of manipulating 3D geometries apart from the WIMP paradigm (Fig2). For example, 3D scanning or reconstruction using photographs, sketch-based systems and parametric modelling.

Art, being an iterative process, often starts off with basic shapes (outlines in 2D and simple geometries in 3D) and are continually refined. This in-between stage where simple shapes are created, though important, could perhaps be somewhat automated. Just like breaking the intimidating blank canvas for 2D artist, perhaps automating this step could provide 3D artists a head start. Furthermore, with traditional CG pipeline, there is a transition from 2D(pre-production) to 3D(modelling). Perhaps something that could bridge this transition is worth exploring?

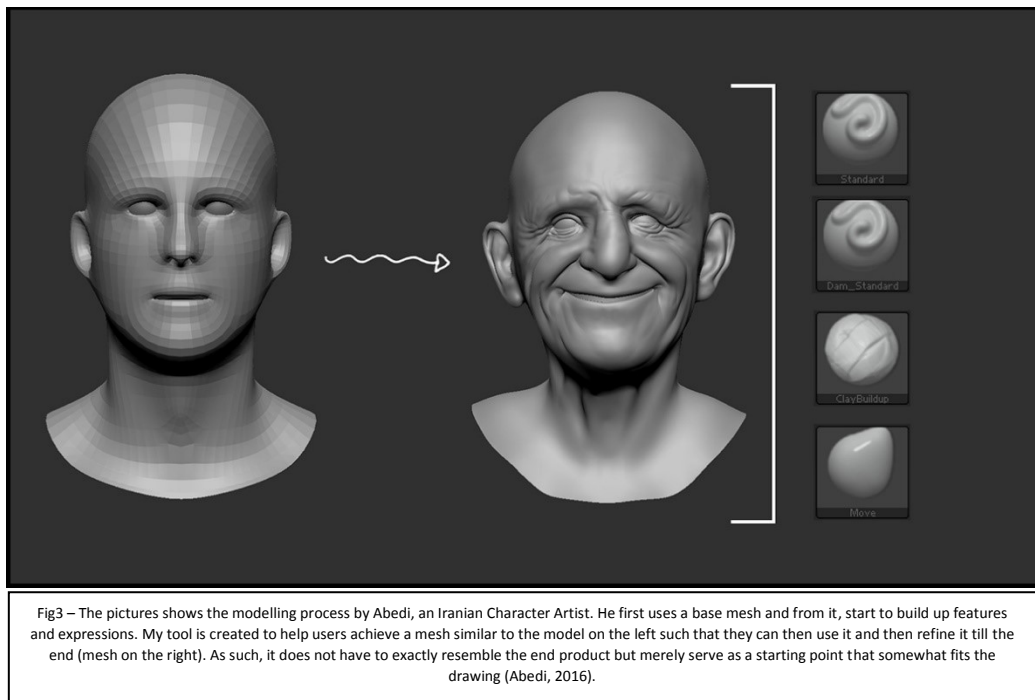


Fig2 – Even back in 1962, modelling systems that is not of the WIMP paradigm, such as Sutherland's sketchpad system has been invented. (Sutherland 1962)

In my innovations, I would like to explore other ways of manipulating 3D geometries apart from the usual WIMP paradigm. With the research I have done, I would like to create an experimental tool in Maya that uses a front and side orthographic drawing, together with a 3D library geometry, to generate a basic head mesh for the artist to continue refining (Fig3). Thus the product is not supposed to fully resemble the end product but only serve as a start for iterative modelling.

The first reason it is innovative is CG allows for the creation of cartoony and expressive geometries. Therefore, instead of the many methods that uses photographs as an input, I would like to explore using drawings instead. Second, I want to create or keep the resulting geometry with animate-able topology. Third, unlike other existing tools which are independent, I want to use Maya as

the main software for my tool, thus having to come up with solutions and work arounds for Maya's toolset and capabilities. Last, I am stepping out of my comfort zone and exploring a different method of approaching a familiar problem. It may not be exactly a new method of modelling or manipulation, but it is an approach that is innovative to me.



### 3D reconstruction using photographs and scanned data

In 1972, Ed Catmull and Fred Parke created *A Computer Animated Hand* (1972) as part of a scientific research project (Utterson, 2011). A plaster model was first casted from a mould of Catmull's left hand and a 3D wireframe was drawn on it. The coordinates were scanned and input into a computer as a series of vectors, creating a CG hand.

Fast forwarding in time, technologies revolving around the idea of scanning and 3D reconstruction have improved leaps and bounds. 3D scanning and reconstruction has advanced to a point where even the minutest of details could be detected - whether with the use of a photo studio or even just a laser and a mobile device (Fig4, 5).

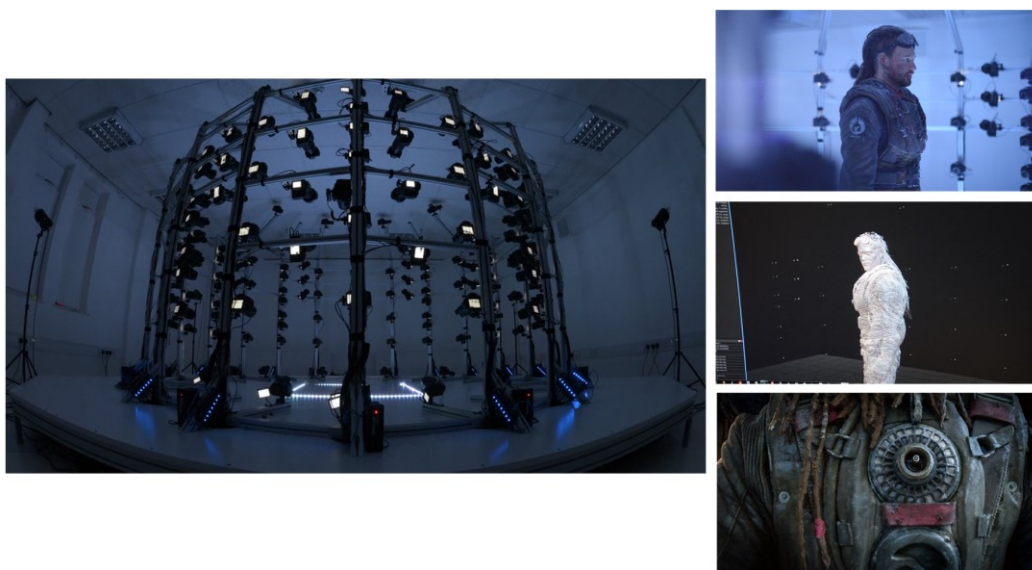


Fig4 – A capture stage and the process of capturing the *Apocalypse* by Ten24 is an example of how much 3D scanning and reconstruction has evolved. (Ten24, 2015)

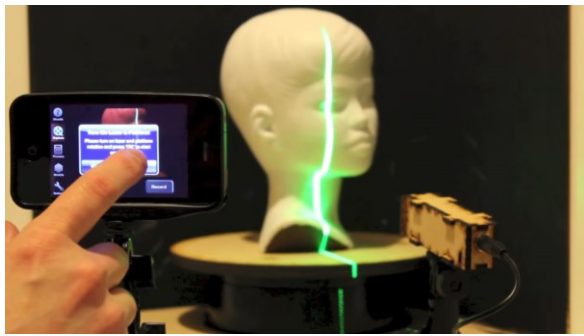


Fig5 – Even with the use of a mobile device and a laser is possible for 3D reconstruction with today's technology. (3Ders 2016)

An approach that uses a database of photographs to reconstruct a location in 3D is explained by Snavely in a Google Tech Talk, *This Distributed Camera* (2012). 2D photographs are first studied to find matching salient points (points that are not completely blank surfaces but have recognisable amount of structural details like corners). Next, the matching points' camera position, orientation and parameters needs to be known. The 2D point is then mapped onto a 3D ray fired from the camera where the point of intersection is the point in 3D space. With access

to a huge database of information, a network graph with some known absolute positions and other relative positions could be constructed and checked to minimise errors.

This process of scanning often results in accurate information as the input data is extensive. However, this huge input data may not be as easily available. Furthermore, as the result is a point cloud, it would be difficult to obtain animate-able topology. Despite that, the process of how a 3D coordinate could be found using 2D images is valuable.

### Sketch-based modelling

*"Everyone can draw" may not be a strictly accurate statement, but there seems to be a universal capacity for visual communication. It is why primitive men told stories through hieroglyphs, and why every meeting room has a whiteboard adorning the wall.'*  
(Olsen, Samavati, Sousa, Jorge, 2009. Pg.85)

SBIM (Sketch-Based Interfaces for Modelling) is motivated by the ease of sketching, as well as how easy it is to communicate information with it. As explained in *Sketch-based modelling: A survey* (2009) by Olsen, Samavati, Sousa and Jorge, it allows for a more natural process of modifying 3D models as compared to WIMP (Fig6).

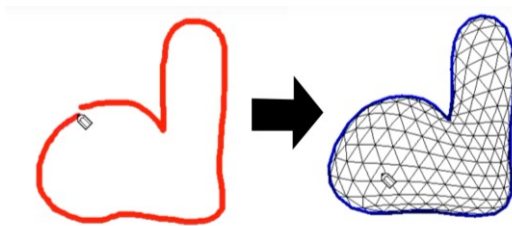


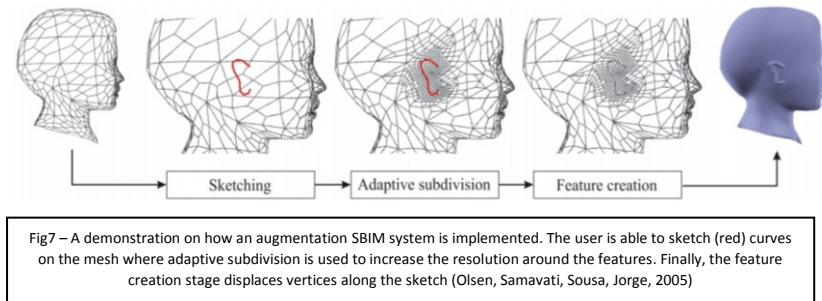
Fig6 – Fibernesh, an example of an SBIM approach. Geometries are created with the idea of sketching and manipulating curves. (Nealen, 2007)

A traditional SBIM approach involves the following stages. A sketch is first acquired by the user. It is filtered to reduce the amount of noisy samples present due to user or device error such as hand jittering or oversketching. This could be done by resampling and approximation algorithms. Next, fitting is done to imbue meaning to the sketch.

Lastly, sketch interpretation is implemented where 2D data is mapped onto a 3D operation. This could, again, be broken down into 3 different modes; creation, augmentation and deformation. (Pg.87-90)

Model creation systems attempts to reconstruct 3D models from 2D inputs. They could either be evocative or constructive. Evocative systems recognise a sketch from a database of templates where it is then used to reconstruct the geometry. Constructive systems forgo the recognition step and

simply try to reconstruct the geometry using various rule-based algorithms. Augmentation systems use the 2D data to create more elaborate details on existing models (Fig7). Deformation systems involve the manipulating of an existing geometry using lattices or deformation algorithms to match the input sketch. (Pg.90-95)



The SBIM thrives on the fact that drawing is easily understood and acquired. However, although being less cumbersome than WIMP, a sketch based result is not predictable and deterministic and requires more guesswork and inference (Pg.99). Furthermore, it is difficult to simulate the vast nature of human visual memory in a computer as it is not exposed to visuals every day. Lastly, models created using the SBIM approach are often blobby and lack detailed features (Pg.100). However, as the process I am trying to achieve is to create a base mesh for a 3D artist to start with, it may be still acceptable.

## Parametric modelling

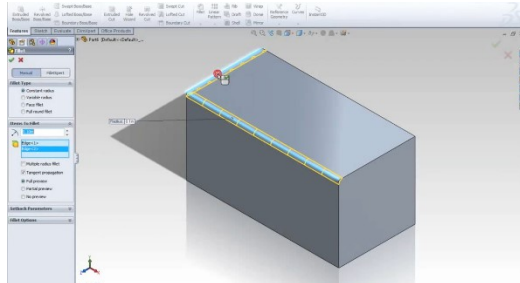


Fig8 – Softwares like SolidWorks, CATIA and Siemens NX utilizes parametric modelling. This diagram showcases Solidworks where the user just clicks an edge and it will create the pattern for them. (GoEngineer, 2011)

Parametric modelling is designed to model component attributes with real world behaviours (DesignTech 2008). It involves heavily on mathematical equations and is based on real life information. The idea of parametric modelling is that if a user were to change a single parameter, other related parameters would get updated automatically (Fig8). The two most popular representation models of parametric modelling are Constructive Solid Geometry (CSG) and Boundary Representation (BR) (Fig9).

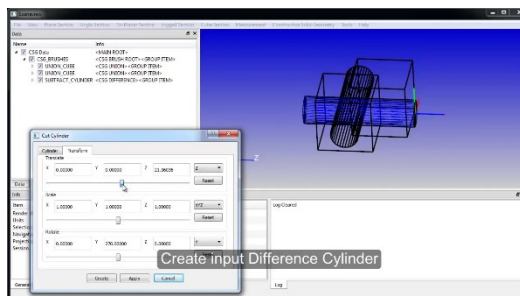


Fig9 – Constructive Solid Geometry (CSG) involves the idea of combining basic geometries together to construct complicated ones using operations such as Boolean and sweeping. Boundary Representation (BR) defines spatial boundaries and join these spatial points to create the object. In this diagram, an example of a CSG modelling technique is demonstrated where the cylinder is used to cut the boxes. (Exanaview, 2015)

Parametric modelling also allows users to create features and embed it into the system, allowing an accumulation of information by building up this library (engineerdotcom, 2012). Although parametric modelling allows for quick design turnaround and increased efficiency, it may be inflexible as it models real world information and it may be hard for an artist to express their creativity with only a few parameters available (engineerdotcom, 2012). However, the ability to expand an existing library could be a solution to the lack of data available in the previous two methods.



## **Related works**

Similar research that also aims to model geometries using orthographic images is proposed by S.K.Singh , M.Kumar , N.Singh and S.Kumar(2014). They use colour coded contours to differentiate the axis of the orthographic images to model solid geometries. The images are transformed according to their axis, and a main axis is chosen with its extrusion length determined by the other planes. The idea of using colour code to imbue meaning to a drawing is similar to my idea of using colour coded lines to differentiate the user's drawing, and this example serves as a good reference.

Another method used to model organic human heads is proposed by Lee and Magnenat-Thalmann (2000). Their method involves creating two 2D wireframes for the front and side view: composed of feature points and predefined relationships with each other. A point is then used from the input orthographic photographs to match these wireframes using various transformation matrices. Next, a semi-automated feature point extraction method with a user-interface is provided. The user can use this interface to make sure the feature points detected are correct. With the data obtained, a correspondence between feature points and control points on a generic model is created for head deformation using a discrete snake model acting on each pixel for colour interest. Finally, the geometry is deformed using non-linear and discontinuous free form deformations. This method is very inspirational and helped me in refining my approach to the tool. The use of a pre-defined relationship with the base mesh and subsequently the user's input helps to avoid assumptions in often ambiguous user-based drawings. As the method involved is only for the creation of a human head from photographs, perhaps functionality that allows users to implement other meshes and have these "pre-defined relationships" created at runtime could be an extension of the tool.

## **Implementation**

### **Model Creation System vs Deformation System**

After some initial experiments (see appendix), I began weighing the pros and cons of a model creation system and a deformation system.

With a model creation system, the orthographic input of the user would be used to generate guide curves where it is used to generate the geometry using surfacing tools such as birail and boundary. With a deformation system, a 3D topologically correct mesh is used and control curves would be created from it. The orthographic input would serve to create guide points where the control curves will move, hence deforming the geometry.

The advantages of a model creation system is that the tool would be more flexible as a model is directly created from the user's drawings. A deformation system has to have a drawing that resembles the library geometry. Furthermore, a model creation system may produce smoother results as primitives are created on the fly, whereas a deformation system could produce unsightly results if a mesh is deformed more than it could handle.

However, as one of the goal is to achieve animate-able topology, it would be difficult to attain it in a model creation system from initial experiments. More guide curves have to be drawn by the user to specify how a mesh should look and how the edges should flow. This is unlike a deformation system where the mesh and topology is already created and some inputs and contours could be assumed. After talking to my peers to see what they would prefer, many said they'd rather spend less time creating the inputs - at the expense of less accurate base geometry since it is only a starting mesh for their modelling process. Therefore, I chose to create the tool using a deformation system while working around possible solutions of its shortcomings.

## Implementation of a library

A deformation system faces a constraint where the input images have to resemble the chosen library. To tackle this, a library system could be implemented where users have the ability to extend this library. This also allows users to share libraries with each other, thus creating a database and improving the tool's capabilities with time.

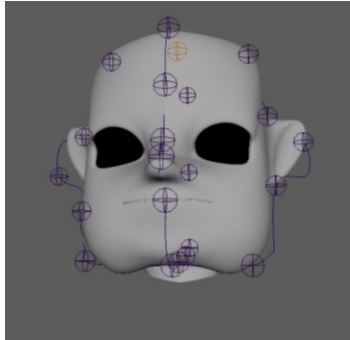


Fig10 – Initially, the mesh was transformed with the help of joints instead of curves. Looking back, perhaps a joint approach instead of a wire deformer may be better. This is because with a joint approach, the weights fade out more naturally and can be anchored by additional joints

## Working with separate mesh parts

The initial tool involves working with a whole mesh and deforming it with joints. This allows, by default, the vertices of the mesh to have a total influence of 1.0 (Fig10). However, mesh that has parts close together may get unintentionally deformed due to the area of influence. A solution to remedy this was to separate out the geometry into parts, deform them and then combine them back together. Wire deformers are used instead of joints as its area of influence is more controlled.

## Differentiating parts in a sketch

Fitting is necessary to analyse the input drawings and convert them into useful information. For this, colour codes are used to represent the various parts of a sketch.

## Creating the controls for a library mesh

The initial idea was for the user to select vertices on the mesh that would represent where each body part would be (Fig11). However, this results in a not-so-friendly user-interface.

Shading groups are then introduced to allow the user to specify a group of faces that is associated to a body part (Fig12). Not only is this more intuitive, it allows for an additional splash of colour.

As proposed earlier, calculating and creating “pre-defined relationships” at runtime could be a good inclusion. This pre-defined relationship exists in the control curves of the head. To create this in runtime, initially, a known vector like an up direction is used to compare against edges. However, this does not take into account slopping vertical edges list a wide

nose bridge. Thus, I

instead separated the head mesh into parts depending on where the shader is applied and classify them as two kinds of shapes. Circular shapes (eyes, mouth) and squarish shapes (cheek, nose, forehead).

For both shaped geometries, the border edges are detected first. For circular shapes, the inner and outer border edges are found by checking their respective bounding boxes. For certain squarish shapes (forehead), the corner vertices are found first and its position in 3D space is queried. With the derived

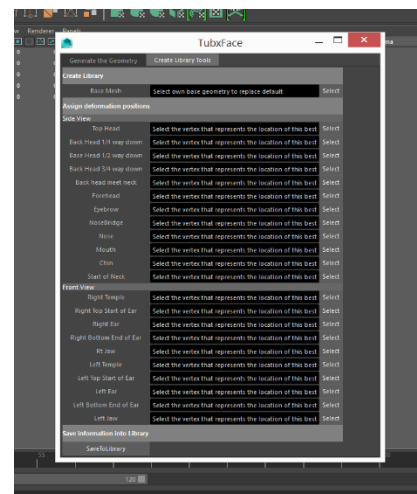


Fig11 - The initial UI where the user manually picks each vertex that they think represents a certain body part. The amount of input is massive. The screenshot only represents the profile curves input.

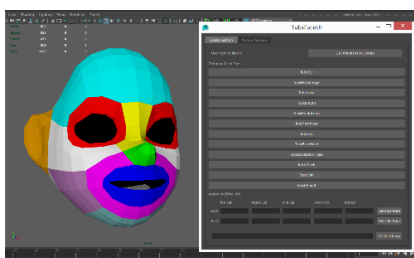


Fig12 - The revised UI. The user picks faces and click on the buttons that corresponds to what the faces are (eyes, mouth etc). It is more colourful and user friendly.

information, the corner vertices are then grouped and pairs of them, if necessary, are connected by finding the shortest edge. For other squarish parts (nosebridge), the border edges are checked first to find where it's connected to. Edges that are not necessary are removed (Fig13).

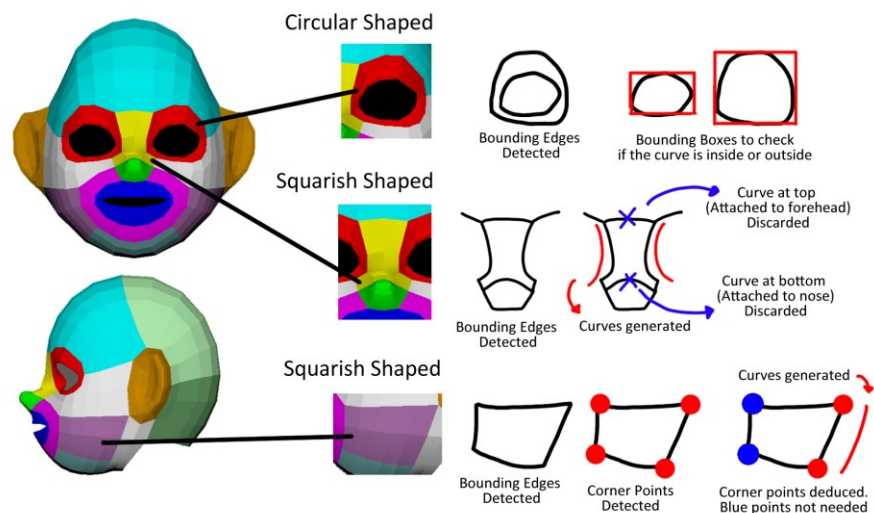


Fig 13 - An illustration of the different ways control curves are generated

### Reading image data in Maya

To read images into Maya, I used the MImage class from the Maya API. However as the Maya API is designed as a C++ library, it uses pointers which python lacks. Instead, the Maya API MScriptUtil class is used as a work around (Fig14).

```
self.image = om.MImage()
```

```
self.ImageWidth = om.MScriptUtil()
self.ImageHeight = om.MScriptUtil()
```

```
WidthPtr = self.ImageWidth.asUIntPtr()
HeightPtr = self.ImageHeight.asUIntPtr()
```

```
self.image.getSize(WidthPtr,HeightPtr)
```

```
self.m_width = self.ImageWidth.getUInt(WidthPtr)
self.m_height = self.ImageHeight.getUInt(HeightPtr)
```

Fig14 - Code snippets showing the use of MScriptUtil to solve the lack of pointers in Python

### Scanning the image

The process of reading images is split into a scanning and a tracing stage.

During the scanning stage, by default, the first pixel of an MImage image is at the bottom left [0,0]. It scans from bottom to top(row) and left to right(column). However, sometimes a scan from another



direction is necessary (Fig15). Thus, the scanning function has to be able to specify how and where to scan.

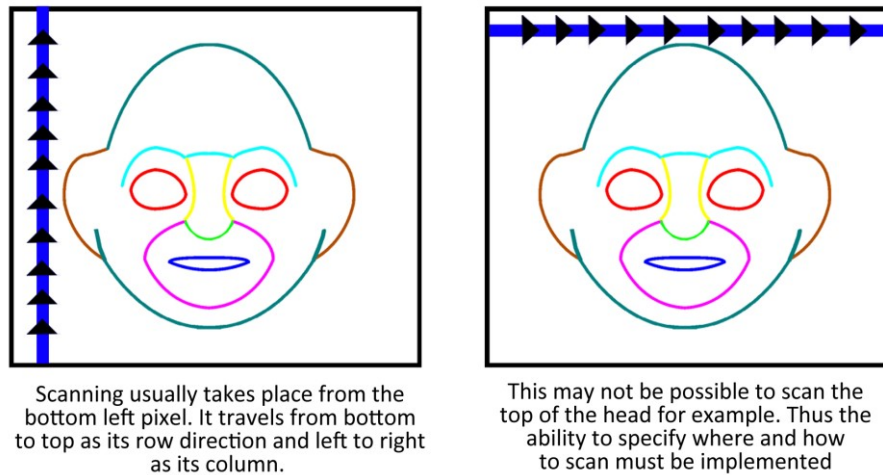


Fig15 - The reason for the need to specify how and where to scan is illustrated in this diagram.

When the desired pixel is detected, the pixel coordinate is passed to the tracing stage. Using a 3x3 kernel, starting from the pixel to the bottom and going anti-clockwise, we check in sequence whether the previous pixel is not of the intended colour but the next pixel is. If it is true, we move (Fig16). However, some situations, as shown in figure 17, could cause an infinite loop. To fix this, I implemented Moore-Neighbour tracing algorithm. The algorithm is almost similar to the previous - with the exception of starting from the pixel which it came from instead (Fig18).

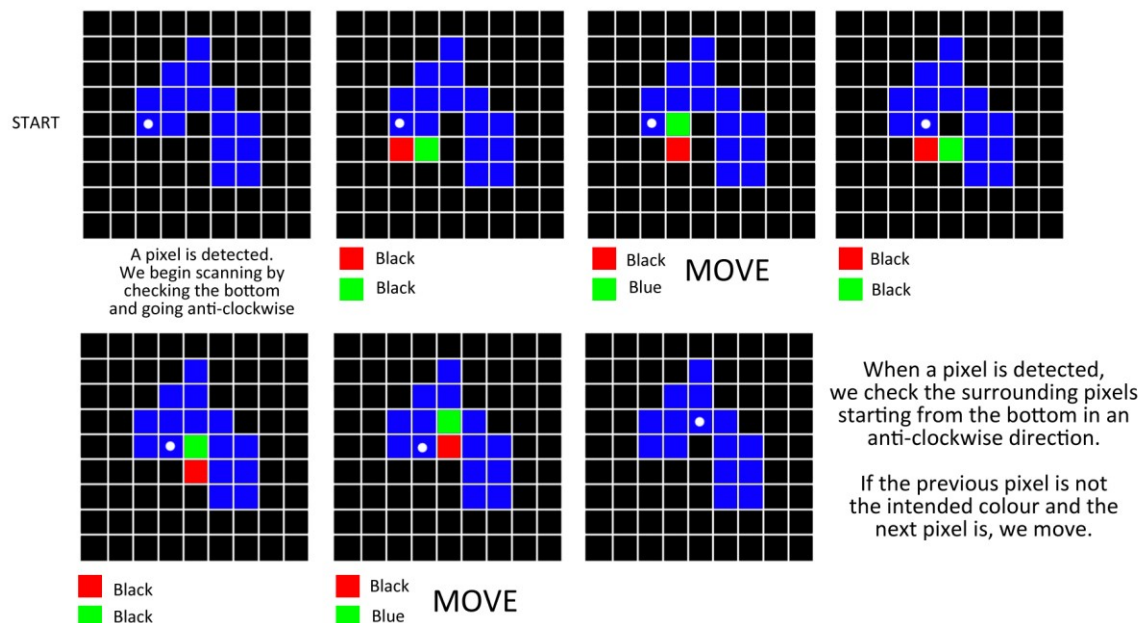


Fig 16 - The initial tracing algorithm where we compare 2 pixels in a sequence.

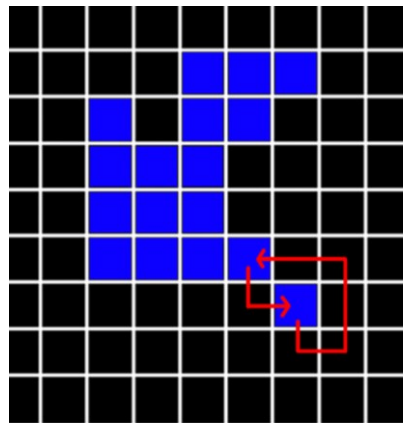


Fig17 - Situations such as what is shown in the diagram may cause an infinite loop as we always start the trace from bottom.

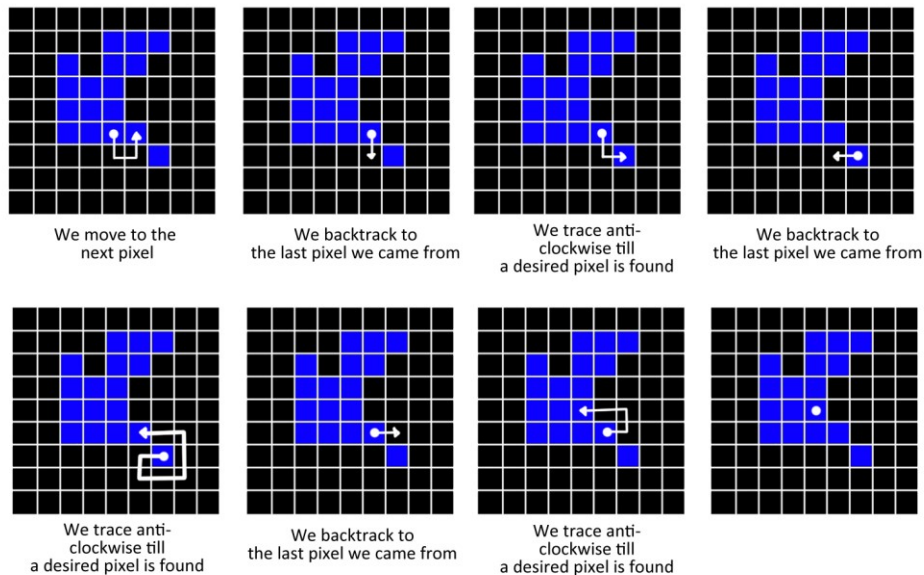


Fig18 - Moore-Neighbour tracing algorithm is applied. Instead of always starting the check from the bottom pixel, we start from the last pixel we came from.

A slight addition to the tracing phase is distinguishing and storing the end most pixel's coordinate according to the direction of scan. This is so the coordinate could be passed back to the scanning stage in order to scan for any more pixels of the desired colour (Fig19). To prevent noise in user's drawings, the front pixel list is filtered and reduce to a pre-defined amount of points at equal spans.

As the scan results in a closed cylinder due to the user using a bigger size brush, there are times where this cylinder is meant to be a line. For such cases, the scan points are rearranged to form the 2 halves of the cylinder where it gets divided.

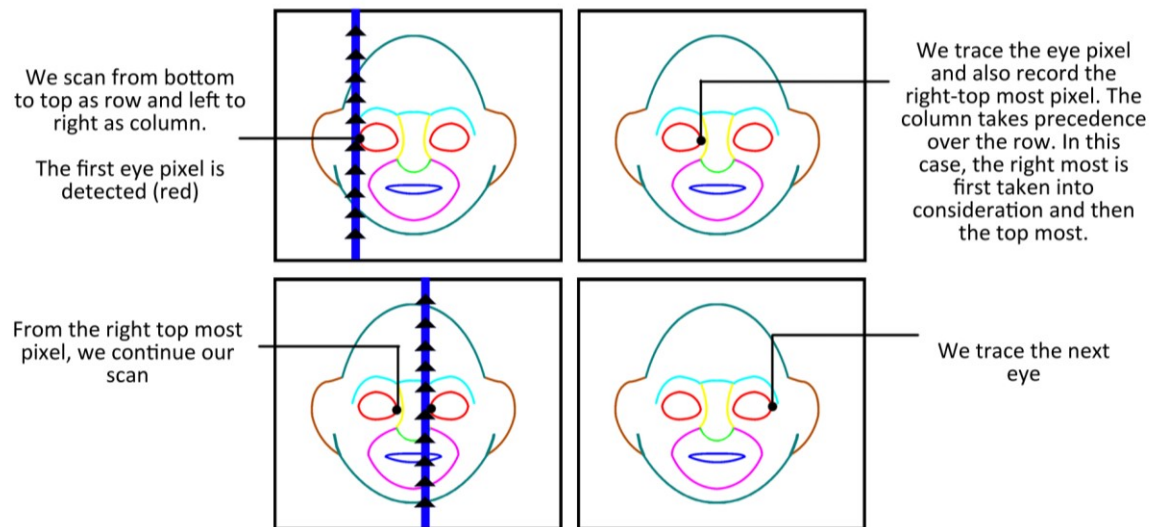
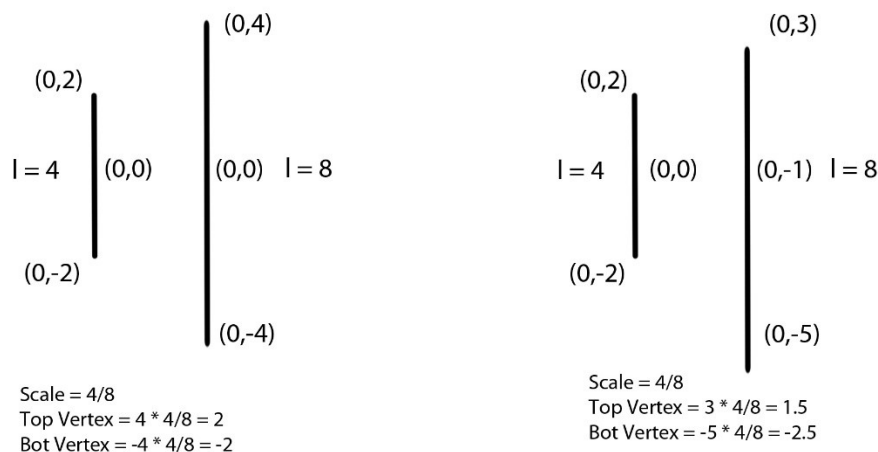


Fig19 - A diagram illustrating how the end most pixel is detected and why we require it.

### Translating 2D pixel data into 3D maya coordinates

During the scanning stage, an additional step of finding a matching pixel from the front and side view is required. This could be done from the way an image is scanned or the rearrangement of the data to start from a similar point.

After getting a set amount of pixels from the front view and a list of pixels from the side, the side view pixels are translated in the y-axis to match the center point of the front view (Fig20). As the front view represents the  $[x,y]$  coordinates and the side, the  $[z,y]$  coordinates, the y values could be checked against each other and the corresponding z value could be obtained. However, for this implementation, the side view pixel range has to be greater than the front view in order for every front pixel to get a corresponding z value (Fig21). Furthermore, as it is a one-to-one correspondence, side views with looping lines pose a problem (Fig22).



To fix this we match the middle offset before scaling

Fig20 - As we can see in this example, the scaling would be different if the center axis is not matched prior to scaling. Even though both examples show lines of equal length, the right example have the top and bottom vertex scaled wrongly.

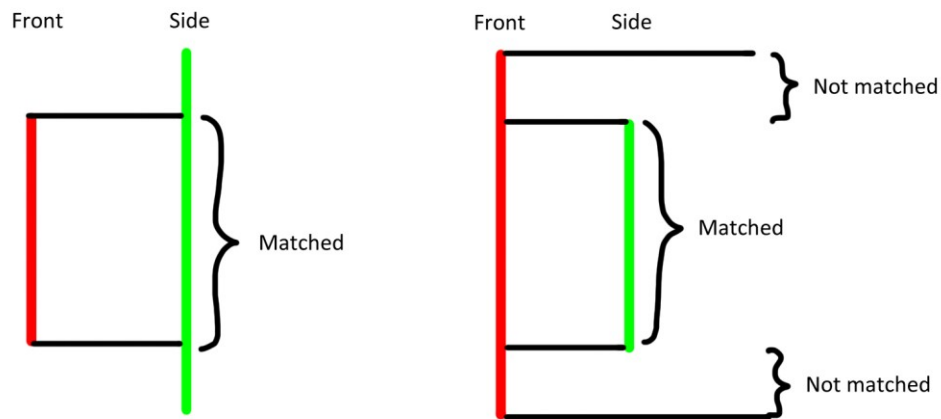


Fig21 – This example shows the reason why a side view needs to be longer than the front. A one to one correspondence to the front view has to be reached. Thus if the side view data is limited, there would be some pixels from the front view that does not have a matching side.

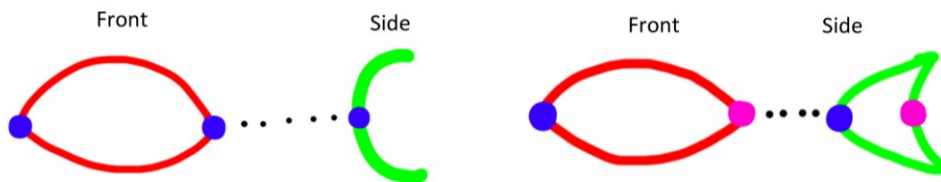


Fig22 – This diagram shows the limitation of the algorithm. At the moment, if a Y value matches in the front and side, the corresponding Z value is taken. However, if one has a side view line that is looped, only the outer part of the side view Z depth is used (Blue Point). What should happen instead is the blue point should match with the side view's blue and the pink point should match with the side view's pink. As a temporary solution, the user is given an option to correct the scanned points before proceeding with deformation.

### Establishing vertex relationships

Initially, a text file (.tubx) was written which allows for data export/import. This allows for the tool to differentiate between its library files. At the time of import, however, the geometry gets recreated and thus its vertex id would be different. To solve this, I took advantage of the fact that the vertex position would still remain the same, so information could then be recreated by creating a correspondence. In the end I chose to return to a simple maya export together with the data written in a .json file as it fits my needs and is more optimized.

As I am working with separate mesh parts, what was once a vertex occupying a coordinate may become 2 or more vertices. When deformation takes place, it may result in a gap. Thus I use the concept of vertex relationship to counter this problem. It allows me to establish a “parent and child” relationship (Fig23) where the parent being a vertex that is directly influenced by controls and a child are just duplicates of the original vertex. After a deformation takes place, the child vertices would find their corresponding parent and attach themselves back.

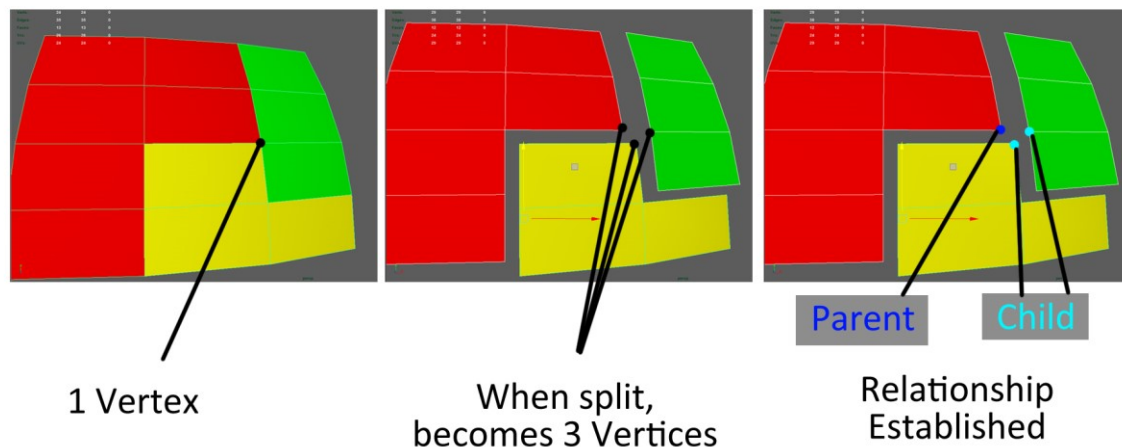


Fig23 – In the diagram, that 1 vertex upon splitting becomes 3. A gap may form as shown when deformation takes place. To remedy that, a relationship is established where one vertex is chosen as a parent and the others a child due to a hierarchy. At the end of the deformations, the child vertex would find and attach itself back to the parent.

### Deforming the geometry

After scanning the images, the 3D data is transformed and scaled down to fit within a 1x1x1 bounding volume. This was to allow pre-defined weight values.

Just like the previous stage, a relationship has to be established between the 3D scanned data from the user's drawings and the control curves. There are 3 such scenerios. The first scenerio involves circular controls whereby an elimination method is used. The data and the controls are determined to be clockwise or anti-clockwise. If they do not match, the data would be flipped. Next, the center points for both data and control is determined and are bubblesorted to the closest X towards the center. The points are halved and are bubblesorted according to ascending Y. Finally, the bottom two points are checked to determine the left most point (Fig24).

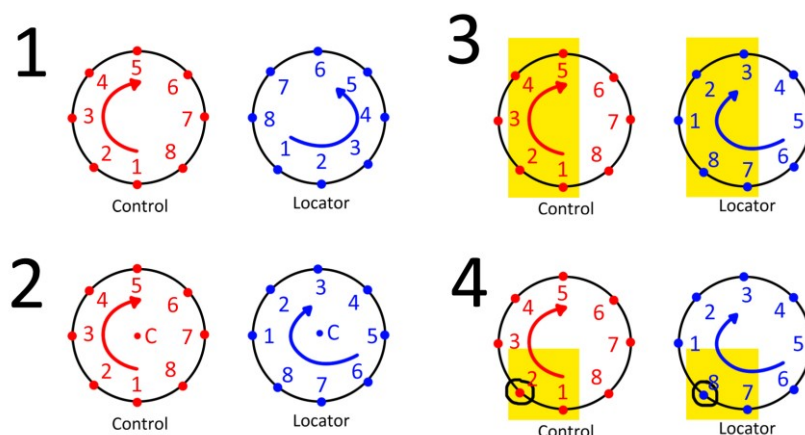


Fig24 – The diagram illustrates how the relationship between control and data is found. In step 1-2, the locator is reversed to match the control direction. From step 2-3, The center point is located and the control and locator are sorted to the closest X against the center. The points are halved and we have the yellow region in step 3. We sort it against Y to get the yellow region in step 4 and we get the left most point.



The second scenerio involves lines. After determining if it is horizontal or verticle, the 2 end points are compared to ascertain its direction. We do the same for the data and we match their direction (Fig25).

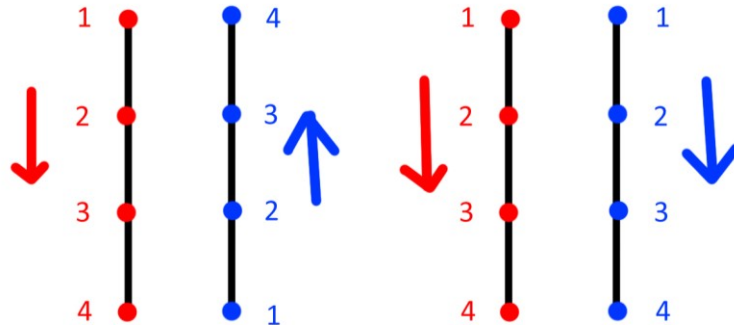


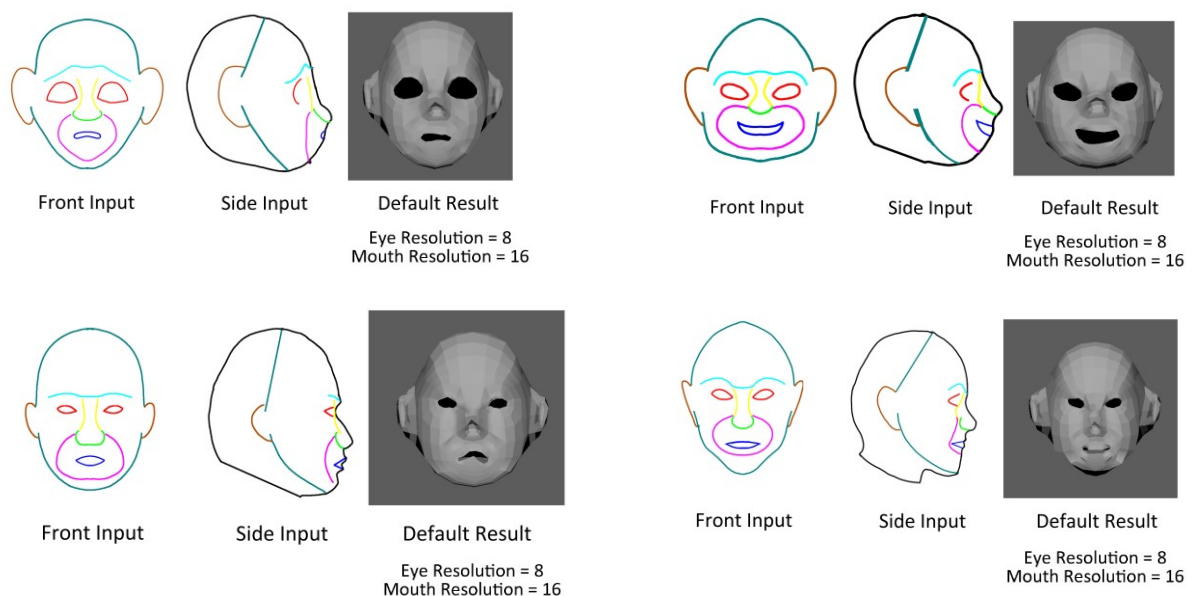
Fig25 – The diagram illustrates how the relationship between control and data is found for the line. The red line represents the control while the blue line represents the locators. As a line travels in one direction, once the line matches in that aspect, the relationship can be created.

The last scenerio is for the profile. As there are less curve points then data in this case, we check which data point a curve point is closest to. We remove that matched data from the list to prevent two curve points from finding the same matching point.

### Clean Up

To reduce the shortcomings of a deformation approach due to over-deformation, a clean up process at the end is used. Inside vertices of the parts are first averaged out before the mesh gets combined. An overall smooth operation is applied to the entire mesh to reduce creases caused by the child-parent finding stage. This however, causes the geometry to loose some of its shapes which is a flaw.

### Results

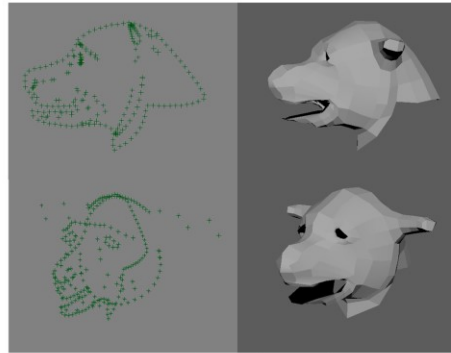




Front Input



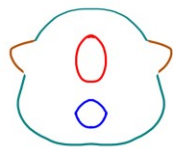
Side Input



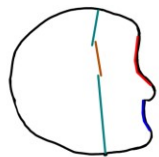
Results with mouth scanned slightly corrected due to scanning limitations

Eye Resolution = 8  
Mouth Resolution = 16

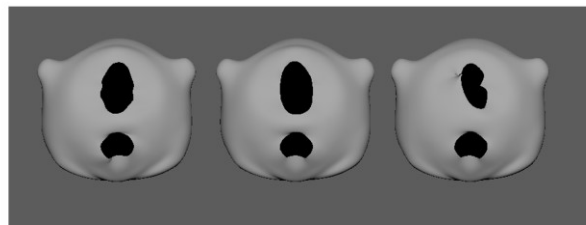
### Default Result



Front Input



Side Input



Eye Resolution = 8  
Mouth Resolution = 16

Eye Resolution = 16  
Mouth Resolution = 24

Eye Resolution = 24  
Mouth Resolution = 32

### Allows Blendshape creation

Front Input



Default Result



Eye Resolution = 16  
Mouth Resolution = 24

## **Conclusion**

Through my innovations, I have researched various ways of manipulating and modelling geometries apart from the WIMP paradigm. I then proceeded to implement a tool that allows the use of a front and side orthographic drawing, together with a selected library geometry, to generate a basic head geometry that keeps its animate-able topology to be used as the starting point for 3D modelling and refinement. The tool could also be used as a pre-production tool where an artist could check how their drawings would look in 3D or a pipeline tool where various artists could divide the job between expanding the library and sculpting the meshes.

## **Solutions to problems highlighted at the start of the project**

To solve the problem of lacking a database and to compensate for the limitations in diversity of a deformation system, an extendable library is incorporated so the tool could be extended with time. To embed information in drawings and the 3D mesh, colour codes are used to differentiate what the data meant is used. For more controlled deformations, the mesh is separated into its various parts and scaled to fit in a common bounding volume. To connect the mesh back, the vertex relationship system establishing a parent-child context is used. Finally, to tackle unsightly results due to overdeformation, vertices are averaged to some degree after deformations.

## **Critical analysis and heading forward on how to improve**

Approaching the end of this assignment, the tool I have produced has not met my expectations. There are currently too many restrictions and constraints in the tool - such as how a user draws their inputs, the disparity between the input drawings and the base geometry, or how the mesh is deformed. There are still possibilities of failures when using the tool incorrectly and this may be hard on the user.

However, I have succeeded in achieving a test version of what I have set out to do. I believe I created innovative solutions and proposals during my workflow and to problems that exist in various study discussions (albeit not being the best or most efficient).

Going forward, I would like to improve my tool's functionality. Over the course of this project, the choice of using a deformation system may have been wrong as currently there are too many limitations over benefits. Perhaps a re-look at a model creation system would be better. However, if I were to stick to a deformation approach, perhaps implementing my own deformation function could allow for better control as the working mechanics of maya's deformers is hidden from the programmer. A function to find the best fit base mesh to the user's drawing could also be implemented with an extensive library.

Matching of a front and side pixel should also be revisited considering the limitations highlighted (Fig21, Fig22). To better its functionality, parametric modelling could also be implemented to change features of the mesh such as its plumpness or race. A randomiser function which creates duplicates of the meshes with varying parameters could also be implemented together to create an option for a crowd modelling tool.

## **What I have learned**

Nevertheless, through this process, I have learned a lot. Aside from the tools, software and modelling approaches, what this process taught me is to never be satisfied with an immediate solution to a given problem. I should always question these solutions to innovate and come up with new ideas. Not only has it allowed me to see a problem in a different manner, it allowed me to understand even more deeply the reasoning of previous implementations I've looked at.

## Reference

- 3Ders, 2013. *Moedls app turns your smartphone into a 3D Scanner* [online]. Available from: <http://www.3ders.org/articles/20130225-moedls-app-turns-your-smartphone-into-a-3d-scanner.html> [Accessed 20 December 2016]
- Abedi, R., 2016. Making of 'Smile at all ages' - 2144\_tid\_image1.jpg [image]. Worcester, UK: 3D Total. Available from : <http://www.3dtotal.com/tutorial/2144-making-of-smile-at-all-ages-maya-v-ray-zbrush-misc-mari-by-reza-abedi-quixel-ddo> [Accessed 5 November 2016]
- Clark, D., 2014. *box-modelling.jpg* [image]. Available from: <https://xxdegallaxx.wordpress.com/2014/02/12/box-modelling/> [Accessed 15 January 2017]
- DesignTech, 2008. *Parametric Modelling* [online]. Available from: <https://www.designtechsys.com/articles/parametric-modelling.php> [Accessed 13 November 2016]
- Engineerdotcom, 2012. *Parametric Modelling – Pros and Cons* [online]. Youtube. Available from: <https://www.youtube.com/watch?v=rRZIMnUf3gA> [Accessed 13 November 2016]
- Exanaview, 2015. *Constructive Solid Geometry Modelling* [online]. Youtube. Available from: <https://www.youtube.com/watch?v=vlDaJigvkqM> [Accessed 19 December 2016]
- Ghuniem,A.G.,2000. *Moore-Neighbour Tracing* [online]. Available from: [http://www.imageprocessingplace.com/downloads\\_V3/root\\_downloads/tutorials/contour\\_tracing\\_Abeer\\_George\\_Ghuneim/moore.html](http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/moore.html) [Accessed 13 January 2017]
- GoEngineer, 2011. *SOLIDWORKS Quick Tip – Getting Started* [online]. Youtube. Available from: <https://www.youtube.com/watch?v=cmC2MLRetko> [Accessed 20 January 2017]
- GoogleTechTalks, 2012. *The Distributed Camera: Modeling the world from Online Photos* [online]. Youtube. Available from: <https://www.youtube.com/watch?v=opRZUMrLJOM> [Accessed 13 November 2016]
- History of Computers, No Date. *Sketchpad of Ivan Sutherland* [online]. Available from: <http://history-computer.com/ModernComputer/Software/Sketchpad.html> [Accessed 5 November 2016]
- Lee, W.S., Magnenat-Thalmann, N., 2000. Fast head modelling for animation: *Image and Vision Computing*. 18, 355-364.
- Liu, S.X., Hu, S.M., Tai, C.L. and Sun, J.G., 2000. A Matrix-Based Approach to Reconstruction of 3D Objects from three orthographic views [online]. *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*. Washington 3-5 October 2000. Hong Kong, China: IEEE Computer Society. 254-261. Available from: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=883948> [Accessed on 2 November 2016]
- Nealen, 2007. *FiberMesh: Designing Freeform Surfaces with 3D Curves* [online]. Youtube. Available from: <https://www.youtube.com/watch?v=W0XGkS7zebo> [Accessed 13 November 2016]
- Olsen, L., Samavati, F., Sousa, M. and Jorge, J., 2005. Sketch-Based Mesh Augmentation: *2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling* [online], 1-10.

Olsen,L., Samavati,F.F., Sousa,M.C. and Jorge,J.A., 2009. Sketch-based modelling: A survey: *Computer & Graphics* [online]. 33, 85-103.

Singh, S.K., Kumar, M., Singh, N. and Kumar, S., 2014. Construction of 3D Model from its orthographic projections using OpenGL: *International Journal of Emerging Technology and Advanced Engineering* [online]. 4(1), 142-153.

Ten 24, 2015. *Capturing the Apocalypse*[video, online]. Vimeo. Available from: <https://vimeo.com/140403848> [Accessed 12 November 2016]

Utterson, A., 2011. *A Computer Animated Hand* [online]. Culpeper, VA: Library of Congress. Available from: [https://www.loc.gov/programs/static/national-film-preservation-board/documents/computer\\_hand2.pdf](https://www.loc.gov/programs/static/national-film-preservation-board/documents/computer_hand2.pdf) [Accessed 13 November 2016]



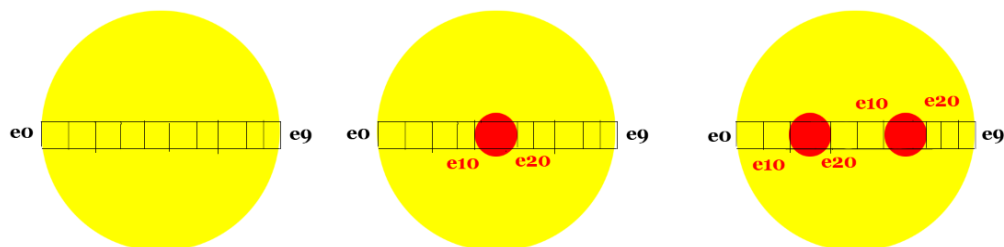
## Appendix

### Initial Research and experimentations

Included in my appendix is a list of initial research and experimentations I did prior to the start of the implementation. At this point, I was experimenting with various methods in order to allow for a more flexible tool.

### Separate mesh deformations

I experimented with the idea of having separate meshes for each body part as my initial thought was that this could perhaps lead to better deformations. This could also allow for customization of parts. For example, a user may include one of his or her own eye mesh as part of the library or perhaps allow a combination of different number of eyes and not be restricted by 2.



Take for example, we have a yellow and a red mesh. With a base mesh already created, we can pre-compute relationships between edges of the polygon. We then create some rules as follow

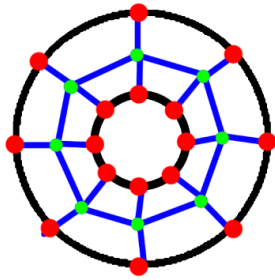
- 1) A yellow mesh at edge e0 has to be connected to edge e9.
- 2) If there is a red mesh, connect it to e10 and exit at e20.

So the procedure would be:

e0 -----> for n number of red meshes, enter e10 and exit e20 -----> e9

However, such is not the case. This is because the number of vertices are fixed for a pre-generated mesh and if we increase the number of eyes, the resulting mesh would increase in the number of vertices. Similarly, if the user incorporates their own body part mesh, it has to have a fixed number of vertices.

### Vertex position calculation



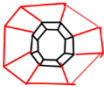
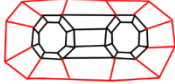
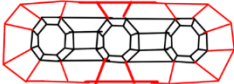
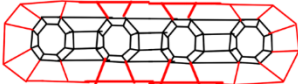
Another way to allow maximum flexibility is calculate every vertex position of a 3D mesh based on the user's input. An additional step of establishing a relationship between each vertex is also needed.

As seen in the diagram on the left, the user's inputs are the black curves. Base on the curve we find 8 points (red dots) and we know that the inner and outer black curves are supposed to be connected. With that said, linear interpolation between the points are calculated (green dots) and the mesh is constructed (blue lines). However, this approach

may be hard to manage as relationship logic has to be created for each vertex in the mesh and it is hard to manage.

### Calculating neighbouring vertices with formulas

I then experimented with patterns to see if there is a way to know how many vertices would there be in a mesh.

	Eye	Loop	Vertices of eye loop
	1	8	8
	2	12	5+5+2
	3	16	5+5+2+4
	4	20	5+5+4+6

So based on the pattern, we can calculate the number of vertices an eye loop should have with  
 $5 + 5 + [(\text{eyes}-2) * 2] + [(\text{eyes}-1) * 2]$   
 $= 5 + 5 + (2 * \text{eyes} - 3) * 2$

Where the first 2 digit (5) represents the number of edges that are connected to the eye loop at the ends when we have 2 or more eyes. This opens the possibility of using formulas to modify the number of vertex in a mesh based on the user's input. However, creating these formula for each body part may be an extremely challenging thing to do. Furthermore, albeit solving the issue of knowing how many vertex there could be, the animate-able topology would be lost and it is stated that it was one of my initial goals.