Final Report

An Easy 3D Modeling Tool

COMP4801 Final year project

Department of Computer Science

Faculty of Engineering

The University of Hong Kong

Supervisor

Professor Francis C.M. Lau

TAI, Tsen-Jung, Airy

3035146669

April 15th, 2018

Abstract

The rapid upgrading of 3D printing technologies currently faces the bottleneck of its incapability of gaining popularity, largely due to the absence of 3D modeling tools which is portable and of low technical requirement while still robust. This report introduces an easy online 3D modeling tool, adopting a block-based modeling style and possessing remarkable deformation functions of skinning surface and digital sculpting. The skinning technique is based on the B-Spline function, which realizes the construction of models with a free-formed side surface. The digital sculpting harnesses intersection test and surface subdivision and applies various types of effects to reform the model surface in detail. The data structures of octree and half-edge are involved for the concern of speed and memory performance. The implementation of surface skinning gives a reasonably good result and several parameters are designed to be adjustable to offer more modeling freedom. However, sculpting remains to demand further research to improve the algorithms of the brush effect as well as the order of intersection test, subdivision and effect application. Additional to the study of algorithms, auxiliary tools and functions are also worthy of attention. A secondary set of a camera and a viewport can be added for better perception during modeling, and decimation can be provided as a complement of subdivision to make progress with the computation performance.

Acknowledgement

I would like to express my sincere appreciation to those who have inspired, encouraged and assisted me during the development of this final year project as well as this completion of report. Special thanks are given to my supervisor Professor Francis C.M. Lau, who has been guiding me from the very beginning towards the accomplishment of this project, and Dr. Bin Chan, who designed and developed the prototype this project based on, and has generously shared his knowledge and expertise with me.

Table of Contents

List of Figures

1. Introduction

1.1 Background

Introduced around 30 years ago, 3D printing has gone through a rapid development with decrease in cost and increase in quality, and it is believed to cast a revolution in fields of industrial manufacturing as well as academic research [1]. However, in contrary to its improvements in technologies, devices and materials, 3D printing suffers a relatively low adoption rate in daily usage [2]. One of the reasons is that although the related hardware and the software can be bought in lower and lower price, the difficulty of creating a customized model for printing remains high [3]. Most of the advanced desktop applications which are used for 3D modeling, such as Blender and ZBrush, have overwhelming user interfaces and provide comprehensive but complex modeling functions, resulting a steep learning curve and obstruct the spread of home 3D printing.

The constrains of these traditional desktop applications have given rise to simplified online 3D modeling platforms. Different platforms have their own characteristics and targeting users due to the differences of their building units and modeling toolkits. SketchUp, as indicated by its name, adopts a sketch-based modeling method by providing users with a sketchbook and toolbox, thus users can start with 2D elements and transform them into 3D using a push-pull handler [4]. However, requiring users to start from scratch, SketchUp is thought to be unfriendly to people who do not have a pre-acquired artistic background [5]. VECTARY offers starting blocks including but not limited to cubes, spheres and cylinders, and allows further modification based on vertices, lines, and triangle meshes [6]. Though this method gives users a stage to begin with, the process is highly time consuming since modeling involves operations to each subgroup of meshes. Tinkercad is another famous online modeling tool with its focus on ease and convenience of simple geometrical modeling. The modeling is accomplished based on regrouping of various basic blocks just like playing with LEGO bricks [7]. Although it is easier and faster to build primary models and less aesthetic sense is required, the cost of time can increase exponentially with the level of complexity of the models. And since the building blocks are predefined and only a few parameters can be manually adjusted, free-transform surface is not available. There are many other kinds of online modeling tools emphasizing on different aspects to release the burden of creating 3D models to some extent. However, the tradeoff between being user-friendly while providing advanced deformation functions remains a problem to solve.

1.2 Objectives

This final year project, an easy 3D modeling tool, purposes a solution to the tradeoff challenge by combining the block-based modeling style with options about free-form surface. The options are as below:

1) The user can create a model with B-Spline surface skinning; and
2) the user can deform the model surface in fine detail using digital sculpting.

And an underlying requirement is that the computational speed and memory consumption are maintained within an acceptable range so that the performance and user experience of the online platform are not sacrificed.

1.3 Prototype

This project is based on a prototype developed and kindly provided by Dr. Chan Bin. The prototype is a web project developed with Microsoft Visual Studio, using JavaScript for the frontend and C# for the backend over ASP.NET MVC framework. It adopts the block-based style as Tinkercad for the advantage of low art skills requirement and offers a lightweight user interface (Figure 1.1).

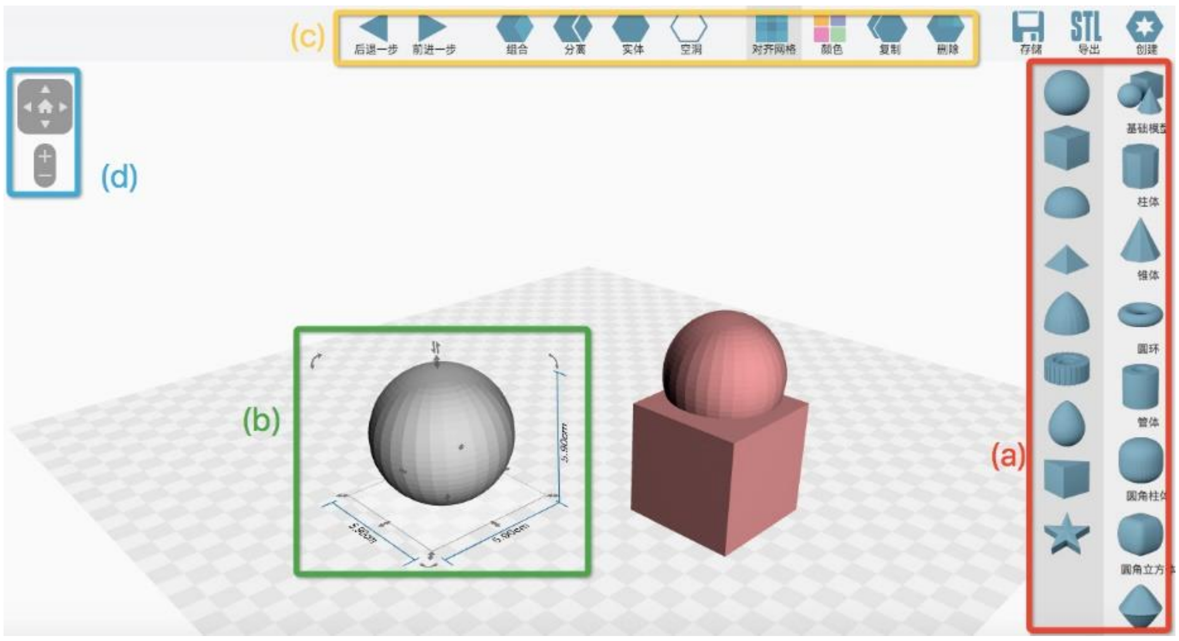


Figure 1.1 A screenshot of the project prototype.

The workspace contains the following:

(a) A bar holding primary building blocks which have no parameter allowed to be adjusted, such as spheres, cubes and pyramids, and a bar holding blocks with changeable parameters, such as prisms or tubes with the number of edges as a variable; and

(b) arrow handlers for scaling, rotating, and transforming a model; and

(c) buttons for commands to operate on one or more targets, such as grouping, making a hole, or duplicating; and

(d) a view controller for changing the camera position.

The prototype has also made its effort towards offering free-form surface. A model with curved side surface can be created by a flat surface swiping along a winding trajectory. Freedom is given to the shape of the flat surface and the trajectory via custom adjustment of uniform cubic B-Spline curves on separate canvases (Figure 1.2).

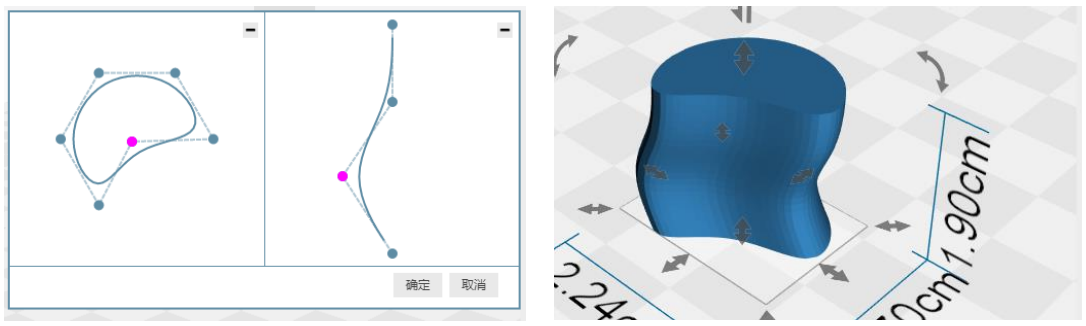


Figure 1.2 Uniform B-Spline curves implemented in the prototype.

The rest of the report begins with descriptions on the methodologies used to accomplish the two objectives: creating models with skinning surface and allowing digital sculpting. Explanations on algorithms chosen and reasoning of choosing criteria are included. The work is divided into 2 phases according to the objectives. Thus, separate sections are used to present the implementation of the user interfaces and logic flows of the two phrases respectively. At the end of the report, there are evaluations on the accomplished outcomes of the two modeling options with suggestions on future work included.

2 Methodology

2.1 B-Spline Curves, Surfaces and Skinning Models

In this project, also as an extension of the prototype, a B-Spline is chosen to represent and generate curved surface. Compared with a Bézier curve, which requires a high degree polynomial to produce a curve with high curvature change, a B-Spline curve is able to use a lower degree to capture a flexible curve, while maintains the G^1 continuity at joint positions of segments. There are two methods to generate a B-Spline curve, by treating the given points as control points or interpolating points respectively. Generally, computing using control points gives a more desirable result since the curve is defined to flow close to the track connected by consecutive points, whilst by interpolation, the curve might be forced to produce crosses in order to ensure every interpolating points is passed by.

Both methods are used in this project to fully utilize the advantages. The flat surfaces which are later used as cross-sections for skinning are created by triangulating closed uniform cubic B-Spline curves. And the side surface is generated by lofting all the cross-sections using interpolation technique [8].

2.1.1 B-Spline Curves Representation and Interpolation

A B-Spline curve is basically decided by a set of control points $\boldsymbol{P}$, a knot vector $\boldsymbol{U}$ and a degree parameter p, thus the data points function $C(u)$ which represents the points on the curve can be denoted as

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P}_i$$

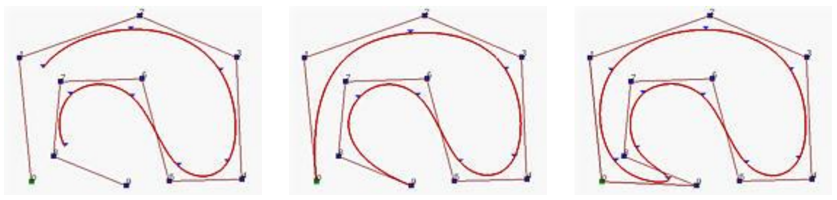
where $N_{i,p}(u)$ are B-spline basis functions defined with degree p, which have a recursive relationship denoted as

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

with a terminating condition

$$N_{i,0}(\mathrm{u}) = \begin{cases} 1, & \text{if } u_i \le u < u_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

showing the feature of local impacts that control points can have on the shape of the curve [9].



(a) Open (b) Clamped (c) Closed

Figure 2.1 Three types of B-Spline Curves.

There are generally three types of B-Spline curves distinguished by the characteristic of being open, clamped or closed (Figure 2.1), and the exact form is controlled by how the knot vector is formed.

Normally, a B-Spline curve is calculated using known control points for the benefit that both the implementation of the construction algorithm and the manipulation of creation can be accomplished with ease. Nevertheless, for the purpose of skinning, it is necessary to perform interpolation, a technique with which a curve or surface can be produced according to given data points lying on the target model.

The definition of the curve interpolation is that, given a set of n+1 data points $\boldsymbol{D}$ and a degree p, a B-Spline curve of the same degree and passing through all the data points in order can be found by obtaining the corresponding set of control points $\boldsymbol{P}$ [9]. The algorithm of the curve interpolation requires auxiliary information about the parameters telling how each data point is fitted (basically, the spacing between neighbor points), and the knot vector. There are diverse methods to calculate the two lists and, in this program, the uniformly spaced method is selected for acquiring the parameters while an "averaging" method is used for the knot vector. Synthesizing the calculated n+1 parameter vector $\boldsymbol{t}$ with the n+1 B-spline basis functions $N_{i,p}(u)$ yields a matrix of $N_{i,p}(\boldsymbol{t}_i)$, and each entry of the matrix can be evaluated using the known data sets of parameters, the knot vector and the degree. Thus a new equation is produced as

$$\mathbf{D}(t_k) = \sum_{i=0}^{n} \boldsymbol{N}_{i,p}(t_k)\boldsymbol{P}_i \qquad for\ 0 \le k \ \le n \qquad \Rightarrow \qquad \mathbf{D} = \mathbf{N} \cdot \mathbf{P}$$

and the control points set can be built by applying a linear system solver [9].

2.1.2 B-Spline Surfaces Representation and Interpolation

A B-Spline surface is constructed by lofting two B-Spline curves, which means that it is defined by a two-dimension sets of control points with m+1 points for its row and n+1 points for its column $\boldsymbol{P}$, and distinct knot vectors $\boldsymbol{U}$ and $\boldsymbol{V}$ and degrees p and q for u and v directions respectively. The mathematics function is presented as

$$S(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}N_{i,p}(u)N_{j,q}(v)\boldsymbol{P}_{i,j}$$

which is actually a tensor-product surface. Similar as the features of B-Spline curves, B-Spline surfaces can also be divided into being open, clamped and closed, and can be produced by providing data points on the surfaces for interpolation.

The process remains alike, starting with selecting 2 sets of interpolation reference parameters $\boldsymbol{s}$ and $\boldsymbol{t}$ for u and v directions, thus the interpolation equation can be formed as

$$\boldsymbol{D}_{cd}=S(s_c,t_d)=\sum_{i=0}^{m}\sum_{j=0}^{n}N_{i,p}(s_c)N_{j,q}(t_d)\boldsymbol{P}_{i,j}$$

and due to the independence between $N_{i,p}(s_c)$ and $N_{j,q}(t_d)$, the equation can be rewritten as

$$\boldsymbol{D}_{cd}=\sum_{i=0}^{m}\boldsymbol{N}_{i,p}(s_c)\boldsymbol{Q}_{id}$$

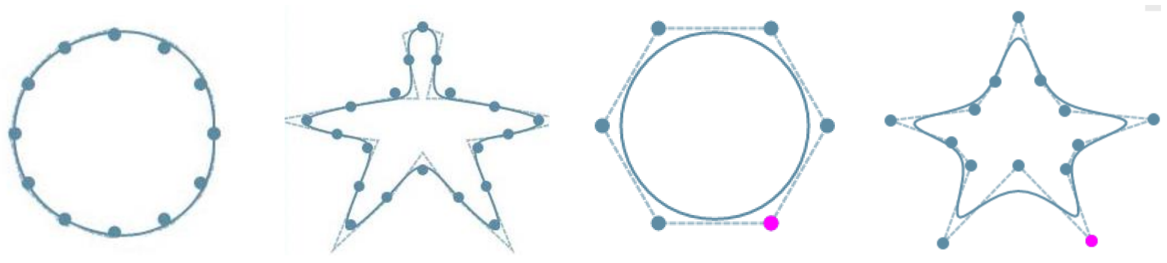
where

$$\boldsymbol{Q}_{id}=\sum_{i=0}^{n}\boldsymbol{N}_{j,q}(t_d)\boldsymbol{P}_{ij}$$

which can be interpreted as the dth column of data points $\boldsymbol{D}_d$ is obtained from the dth column of intermediate points $\boldsymbol{Q}_d$ and parameters s_c. This is again a curve interpolation problem and so as calculating $\boldsymbol{P}_{ij}$ from $\boldsymbol{Q}_{id}$. And the algorithm for solving the points set P turns out to be performing curve interpolation twice by inputting appropriate data sets [9].

2.1.3 Implementation Concern

The decision of using control points to create cross-sections while using interpolating points to build skinning surface is made under the consideration of being user-friendly and intuitive. Since the resulting curve of interpolation can be unpredictable, more input points are demanded for better accuracy, which creates extra workload on users to add and adjust points one by one (Figure 2.2). However, similar shapes can be produced with less control points.



(a) By interpolation. (b) By control points.

Figure 2.2 Similar shapes created using (a) interpolation and (b) control points.

On the contrary, given that the skinning side surface is generated automatically by the algorithm and requires no manual work, interpolation is chosen for skinning so that the model's cross-sections are gradually changed from one to another, following exactly the curves shown on each page of the canvas. This provides users with a better prediction of what is to be generated by modifying the cross-sections.

Basic surface interpolation requires the same number of interpolating points on each cross section. Thus, a layer of interpolating points is selected from each uniform cubic B-Spline curve at certain interval. The intervals are chosen according to the length of each curve so that the selected points on each curve are approximately uniformly spread, while they are of the same number.

2.2 Digital Sculpting

Digital sculpting is the technique to deform the object surface as if it is made of clay. It changes the surface by applying force to stretch triangle meshes or move vertices. In this project, a brush tool is provided to perform sculpting. Different kinds of intersection test are implemented to determine the vertices to be modified. However, the meshes of some primary models lack the sufficient complexity to support sculpting. The problem is tackled using mesh subdivision. The requirements of the modelling tool to operate on advanced 3D graphical features whiling sitting online raise emphasis on the optimization of speed and memory consumption. As a result, data structures of octree and half-edge are implemented for fast intersection test and subdivision. And for saving memory space, incremental subdivision is applied to a selected region of the surface rather than dividing the entire surface into a limit one.

2.2.1 Different Effects Algorithms

The various effects that can be applied to vertices lend strong deformation power to digital sculpting. The effects include but are not limited to drawing, inflating, smoothing, positive or negative pinching and grabbing. The algorithms of realizing each of them are similar with respective specific modification.

The drawing effect is accomplished by moving each active vertex along the same direction of the average normal of all the vertices, while inflating moving each vertex along its own normal. Smoothing is to calculate the average displacement of each vertex to its neighboring vertices and apply the value to the vertex. Pinching is to pull the vertices towards or away from the brush center decided by a user-selected positive or negative mode. Grabbing causes the most dramatic reformation, and distinct to the former effects in which the active vertices keep changing with the new position of the brush tip, those vertices to be affected is defined only once at the mouse-down event, and the mouse moving trajectory is used as the direction to produce new vertices and meshes all the way along [10].

2.2.2 Intersection Test

The process of finding vertices within the sculpting brush region can be divided into the following steps:

1) find the model intersects with the mouse position on the 2D viewport; then
2) find the closest triangle mesh on the model that is hit by the ray cast from the mouse position along the optical axis of the camera; then
3) create a circular brush region according to the radius defined by users; then
4) compute the distance between vertices or triangle faces to the intersection point and compare the distance with the radius to determine whether a vertex or a face is inside or intersect with the brush region.

a) Unproject

The traditional way of intersection test requires transforming the points on the 2D viewport to the 3D world space, which is called unproject by applying the inverse of the camera projection matrix. The casting ray is determined by unprojecting the mouse coordinates to the nearest and the farthest planes of the perspective view volume.

b) Picking

A straightforward way to determine which model is hit by the mouse ray is to loop through all the models in the scene and calculate whether at least one of the meshes is intersect with the ray. This is extremely expensive if there are many models or there are models containing a large number of faces. And it is a waste to take meshes facing backward into computation. A technique called picking provides a powerful solution towards this problem. Picking relies on the shader program which renders the scene and projects the rendered result onto the viewport. When performing picking, the shader program first computes a unique color value for each model using corresponding model ids, then render the models with these distinct colors. When a mouse click event is triggered, the color of the pixel clicked is analyzed to recover the model id information. Since WebGL provides automatic depth test, meshes which are covered by others will not be examined, and this further saves the computation steps.

Although picking can also be applied to determine which face of models is hit by the ray cast, the consumption of memory space increases dramatically since the number of faces is generally much larger than the number of models, and assigning different colors to each face requires a large number of memory allocation. Besides, the ultimate objective of the intersection test is to collect the vertices, which is hard to accomplished by picking. However, looping through all the meshes of a model is still undesired considering that the test is applied repeatedly during the whole process of mouse moving and sculpting.

c) Octree

Octree is a tree data structure allows fast intersection test to an intensive mesh while still being memory efficient. It recursively divides the bounding box space of a mesh into eight child cubes until some termination criteria are met. In this project, the octree of the model is constructed so that [12]:

1) the maximum depth of the tree is no more than 8; and

2) the minimum size of the edge of the cube is no smaller than 0.4 (unit: cm as default); and
3) each leaf cube contains no more than one 100 faces.

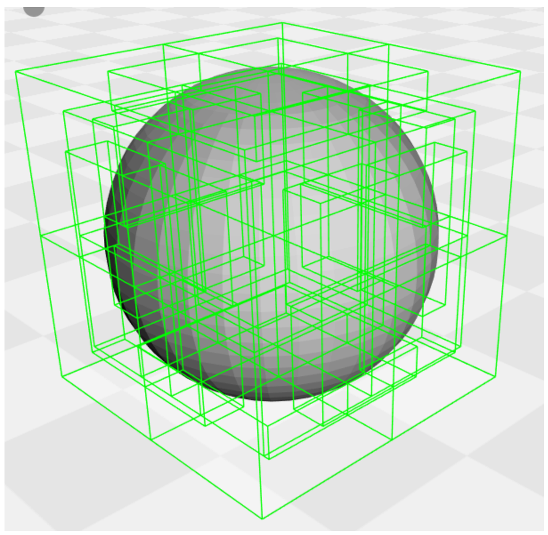


Figure 2.3 An octree “strict” bounding box of a sphere.

These three criteria are maintained in decreasing order of priority and an example of an octree of sphere is shown in Figure 2.3. The faces are sorted into different leaf cubes by comparing their center positions with the cubes’ boundaries. This is referred to as “strict” bounding boxes of leaf cubes. “Loose” bounding boxes are determined by the maximum and minimum coordinate values (x, y, z) of all the faces inside the cubes and are used for intersection test. When do ray-face intersection test, ray- bounding box intersection is performed first and the faces contained in a leaf cube become the candidate faces. Then the candidate faces are tested one by one using modified Möller-Trumbore intersection algorithm [13]. Using the octree structure greatly reduces the number of triangle meshes that we need to involve in intersection computation and improves the processing speed.

The intersection point along with the brush radius defines a round region where any vertex sits inside will be influenced by the sculpting effect. Since vertices can be recovered from face information, checking whether a face intersects with the brush sphere can gather the candidates of vertices which has the potential to lie inside the brush region. It seems that the octree intersection test can also be used when determining faces intersecting with the brush sphere by iteratively examining the candidate faces. However, considering a case where the intersection point is near the boundary of a leaf cube, and unfortunately there is a triangle which has one of its edge inside the brush circle while its center is outside the boundary box, although the triangle definitely intersects with the brush, it is not found since it is not collected as a candidate as the very first stage (Figure 2.4).

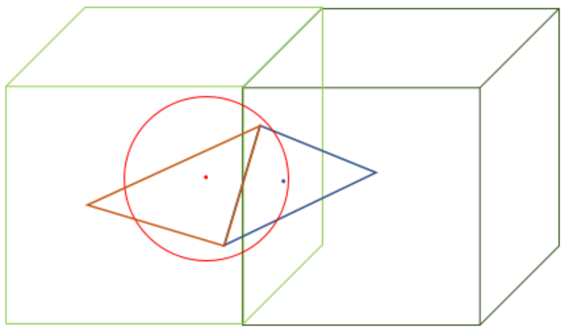


Figure 2.4 The blue triangle intersects with the brush region, but it cannot be found using octree algorithm. Red circle is the region defined by the intersection points. The cube in light green is the leaf cube containing the candidate triangles. The cube in dark green contains the testing triangle.

This might not be a significant problem when the purpose is to collect vertices inside the sphere, since there is probably another triangle mesh bordered by the edge inside the brush and has its center within the intersected leaf cube. Thus, the two vertices of that edge can still be collected through the other triangle. However, problems occur when it is concerning recovering faces, which is required in subdivision process.

The challenge can be solved with the aid of another data structure named half-edge. Half-edge allows quick search of neighbor vertices of a vertex and faces participated by that vertex. The face intersects with the mouse position can be used as the starting argument to collect its neighbors, and by recursively testing neighbor faces' intersection condition, a complete set of faces intersecting with the brush can be recovered. In fact, half-edge is an important data structure to realize subdivision.

2.2.3 Subdivision

Since the basic unit of the effect of digital sculpting is a vertex, the performance is limited by the density of triangular meshes. With a model of a low mesh density, moving a single vertex influences a large part of the surface and fails to accomplish detailed shaping (Figure 2.5). The result gets better with the increase of the number of meshes.

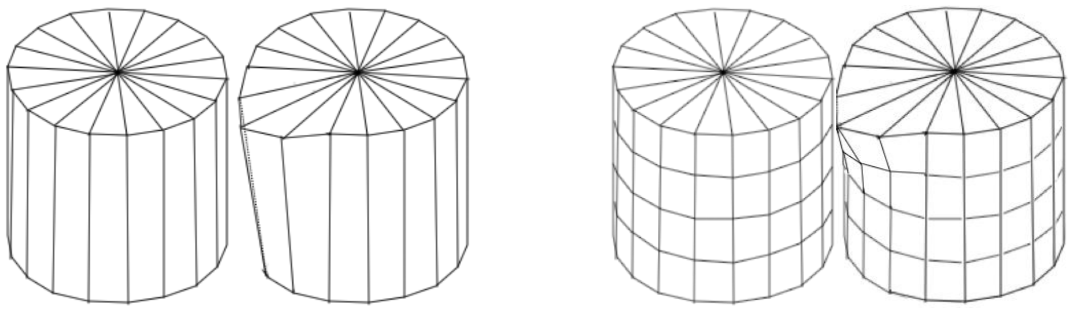


Figure 2.5 Primary building blocks of different polygon density and corresponding sculpting results.

However, in this project, most of the primary building blocks only use a small number of polygons. And it is a waste of memory space and adding meaningless workload of computation due to the maintenance excessive vertices than needed to satisfy the sculpting need which might only occur locally.

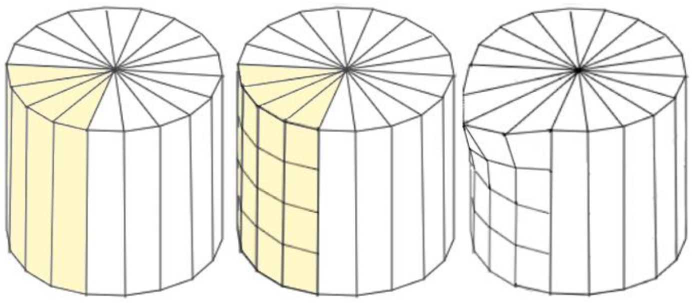


Figure 2.6 Dynamic sculpting in which subdivision occurs only when needed.

Dynamic operator sculpting [14] could be the trade-off between achieving the technique without depleting the data processing capability, in which further-detailed subdivision of meshes is performed only in the area selected by the sculpting brush (Figure 2.6). This reduces the total vertices that need to be stored but preserves the quality of sculpting.

The incremental subdivision for triangle meshes adopted in this project divides the selected region until it has the reasonably good approximation of the limit surface, which is the surface produced if the entire model mesh is divided. This is achieved by recursively dividing the faces inside the selected region until some termination condition is met. The termination condition is when the distance between current position and the limit surface position of a vertex is lower than a predefined threshold. Since not the entire closed mesh is passed into subdivision process, cracks can be produced if adjacent faces are not of same subdivision depth. This is dealt with simple triangulation.

This incremental subdivision algorithm contains the following stages [15]:

1) collect faces intersecting with the sculpting brush; then
2) compute whether the termination condition is met; if not, then
3) get the r-ring neighborhood of the candidate faces; then
4) subdivide the faces using loop subdivision; then
5) remove the cracks using simple triangulation; then
6) repeat stages 2) to 5).

a) Half-Edge

The half-edge data structure relates vertices, edges and faces together to make it convenient to find neighbour information (Figure 2.7). Basically, a half-edge is a directed edge which can be used to traverse within the same face or among different faces. Each half-edge records a next half-edge which sits inside the same face and records a paired half-edge which borders the adjacent face of it. It also records the ending vertex (the vertex the directed edge points to) and the face it contributes to. Thus, given a half-edge, we can easily recover the face it belongs to, the edges and vertices which construct the same face, and the information of its adjacent face. Traditionally, we also relate a vertex to a half-edge which goes out from it. This simplifies the work of finding all the neighbour vertices of a given vertex. Gathering neighbourhood information is necessary in loop subdivision to compute newly added vertices as well as update the position of existing vertices.

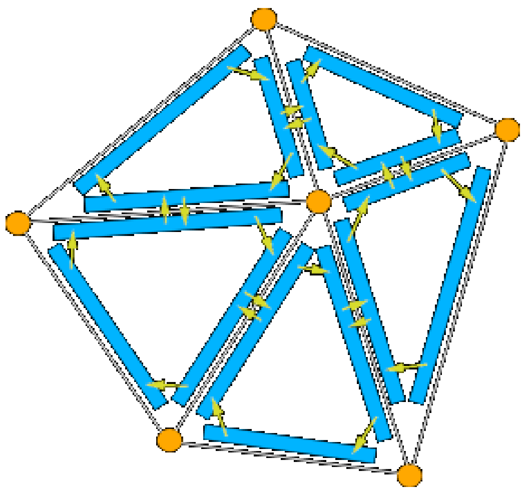


Figure 2.7 The half-edge data structure [16].

b) Loop Subdivision

Loop subdivision updates existing vertices while adding new vertices on edges. In this report, the existing vertices are referred to as even vertices and the newly added vertices are referred to as odd vertices. The new position of an even vertex is calculated using weighted sum of the positions of its neighbour vertices and itself (Figure 2.8(a)) as

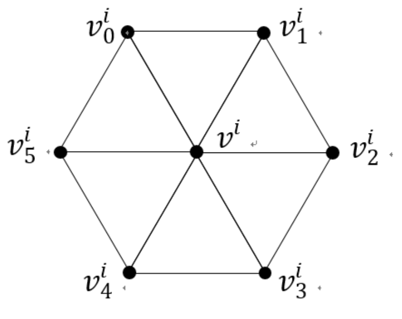
$$v^{i+1} = \beta v^i + \alpha \sum_{j=0}^{n-1} v_j^i$$

where n is the number of the neighbours of vertex v^i and

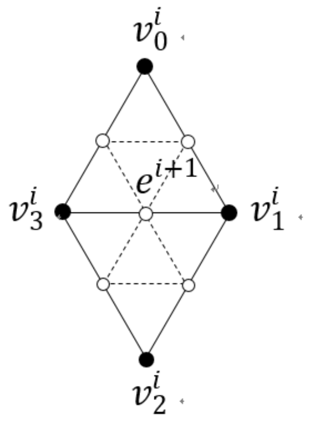
$$\alpha = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\left(\frac{2\pi}{n}\right)\right)^2\right)$$

and

$$\beta = 1 - n\alpha.$$



(a) The new position of the center even vertex after subdivision depends on all its neighbours.



(b) Odd vertices (white) are calculated per edge using the vertices of the adjacent faces (black).

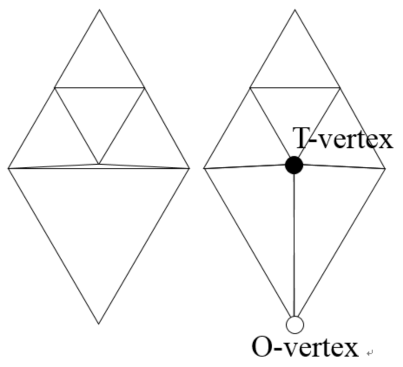
Figure 2.8

An odd vertex is generated by each edge during subdivision (Figure 2.8(b)). And it is calculated using the 4 vertices bordering the 2 adjacent faces of that edge as

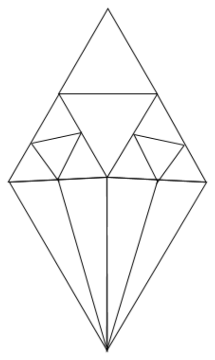
$$e^{i+1} = \frac{1}{8}v_0^i + \frac{3}{8}v_1^i + \frac{1}{8}v_2^i + \frac{3}{8}v_3^i.$$

c) Simple Triangulation

To fill the crack generated between faces of different subdivision depths, we split the face of less depth into halves, and connect the opposite vertex of the split edge (O-vertex) to the newly generated odd vertex (T-vertex) (Figure 2.9(a)). Although by simple triangulation, the crack can be efficiently removed, as we recursively apply subdivision to the selected region, the valence of the O-vertex can increase exponentially and result in undesired ripples (Figure 2.9(b)).



(a) Simple triangulation.



(b) Recursive subdivision resulting ripples at O-Vertex.

Figure 2.9

d) R-Ring Neighbourhood

Collecting r-ring neighbours during each subdivision iteration is raised to solve the ripple problem. A face is said to be at the r-ring region of a set of selected faces if the maximum distance of any of its vertex to the boundary of the selected region is equal to r. As shown in Figure 2.10, the center white face with red vertices is the only selected face, the first coloured layer (orange) is the 1-ring neighbour region, while the second coloured layer (blue) is the 2-ring neighbour region. The red sick border line segments are where simple triangulation takes place with the yellow points as O-vertices.

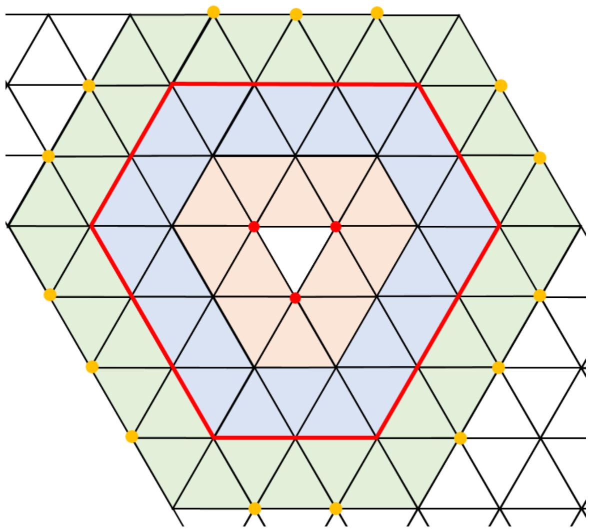


Figure 2.10 R-ring neighbourhood of a selected region.

When performing subdivision, we add r-ring neighbourhood faces into the selected region. The boundary of r-ring neighbourhood shrinks after each iteration, meaning O-vertices keep changing during the process, avoiding adding excessive edges onto the same vertex.

2.2.4 Implementation Concern

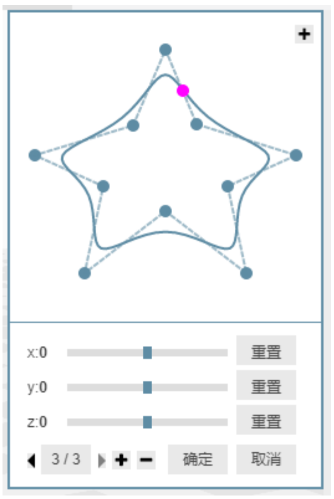
Vertices are modified during subdivision as well as sculpting and since two different data structures are used in this project for convenient computation of different applications, extra work is needed to update the octree and half-edges as soon as the modification of vertices completes. In order not to rebuild both structures every time, during the subdivision and sculpting process, it is important to keep containers to record new and dirty vertices and faces.

3 Phrase One: Surface Skinning

3.1 User Interface

The program provides sufficient while not excessive adjustable parameters which allow users to produce irregular geometry for modelling skinning surface. Users can create a distinguished shape for each cross-section and the number of cross-sections can be changed to satisfy the design need. Each cross-section can have a certain amount of displacement comparing with the bottom surface which is the cross-section defined on the first page.

When users hit the button for skinning models, a model with default settings will be displayed in the scene along with the control panel (Figure 3.1). The control canvas holds a sequence of pages of cross sections which are generated using control points following uniform cubic B-Spline algorithm. On each page, a control point can be moved by dragging it inside the canvas defined area and can be deleted by selecting it and pressing the minus button on the top-right corner. A new control point can be added by hitting any position on the dash line segments and pressing the plus button. The active point either to be deleted or added is colored pink. The cross-section pages can be flipped using the arrow handlers at the bottom of the canvas. A cross section can be deleted by navigating to the corresponding holding page and pressing the minus button next to the arrow handlers, and a new cross section inheriting the same configuration as the current selected page can be added by pressing the plus button. Each cross section can be moved within some range of x, y, z coordinates compared with the bottom surface.



(a) The canvas for adjustment.

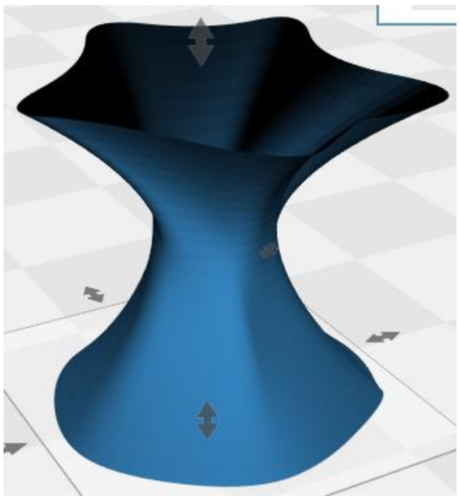

(b) The generated model.

Figure 3.1 The user interface of skinning models.

3.2 Implementation Flow

To generate a skinning side surface model, a list of control points and a list of the displacement vector on each canvas page are passed as inputs to compute a side surface which contains arrays of vertices and indices as an output. It is notable that the number of control points on each page can be different. However, the interpolation points selected from each curve of the cross-section must be of the same number, so that the set of *m*n* points can be used for surface interpolation, in which case m equals to the number of pages and n is the number of interpolation points on each page. A selection algorithm needs to be designed to pick the same number of points which are sit on the curve at an approximately uniform interval. Additionally, the points on the B-Spline curves are of 2-dimension, while when interpolating a surface, 3 dimensional points are needed. A conversion for a point from 2D to 3D is required and related information is provided by the displacement vector.

Using interpolation method to generate the B-Spline surface, basically, is to compute control points set from those interpolation points first and in turn use the set of control points to generate the surface. To recover control points, a parameter selection mechanism need to be defined so that the recovery algorithm can fix the position of a point on the curve by referring to the selection parameter. And the knot vector can correspondingly be calculated. Given a set of *m*n* data points along with predefined B-Spline surface degrees *(p, q)* in *(u,v)* direction, by computing 2 sets of the parameter selections and the knot vectors, a set of *m*n* control points can therefore be recovered and define a surface passing all the initial given data points.

The process to compute 3D interpolation points from 2D control points consists of the following tasks:

1) compute B-Spline functions for each set of control points; then
2) partition *u* from 0 to 1 uniformly into each B-Spline function to generate segment points which can be connected to draw the curve; then
3) compute the number of interpolation points to be used to generate the side surface by checking for the maximum number of control points used among all the pages; use the maximum as the interpolation number; then
4) for each curve, determine an interval by dividing the number of segment points on that curve with the interpolation number; collect segment points according to the interval; then

5) for each layer of selected interpolation points, convert each point using the displacement vector of that layer and the layer index and as a result, a 3D point is produced from the original 2D point.

At this stage, a list of 3D interpolation points for each cross-section is computed and the lists are combined to gain the *m*n* points set. The following steps recover the set of control points which later define the skinning surface from the interpolation points set:

6) determine 2 sets of appropriate parameter selection mechanisms and knot vectors for *u* and *v* directions; in this project, uniformly spaced method is used for parameter selection and the knot vector is generated using a method suggested by de Boor which is to average the parameters [9]; then
7) for a direction of *u* or *v*, a basis function matrix ***N*** is evaluated with each of its element computed using a "fan-out" triangular form [9], and a curve interpolation is performed to get the set of "intermediate data points **Q**, which is then used for another iteration of curve interpolation along the other direction *v* or *u*; After the second round of interpolation, a desired set of *m*n* control points can be obtained.

With the set of control points, the surface mathematical representation can be obtained and like how the cubic B-Spline curves are produced, partitions along *u* and *v* directions are performed to gain segment points within the surface. This is accomplished using de Boor's Algorithm [9]. And once the intermediate segment points are calculated, triangulation can be performed between consecutive layers to draw the surface. The steps are supplied below for the completeness of the entire algorithm:

8) apply de Boor algorithm to obtain segment points sitting on the surface; the segment point is generated by repeatedly inserting a knot until the corresponding multiplicity reaches to the degree number; then
9) triangulate the set of segment points by creating 2 triangle faces for every 4 points on consecutive layers with each layer provides 2 of the points and collect the face indices in counter-clockwise order.

This concludes the generation of a skinning side surface using input arguments of control points and a displacement vector from the adjustment canvas. The returned object is an instance of skinning surface with an array recording the 3D vectors of space positions of all vertices and an array holding the indices sequences for all the faces of the surface.

4 Phrase Two: Digital Sculpting

4.1 User Interface

In this project, the brush radius and effect are the only two adjustable parameters in sculpting option. A scale bar is provided for the radius input and a dropdown menu is provided for effect selection. Taking common user practice into consideration and for the purpose of simplifying the implementation logic, if there are more than one model sitting in selected model sequence when the sculpting toggle is enabled, a grouping command will be operated over these selected models prior to any of functions executed by sculpting. In other words, only one model is allowed to be sculpted at one time. This implementation design is adopted to reduce the complexity of octree and half-edge data structures' maintenance and the interaction between data structures and the subdivision algorithm.

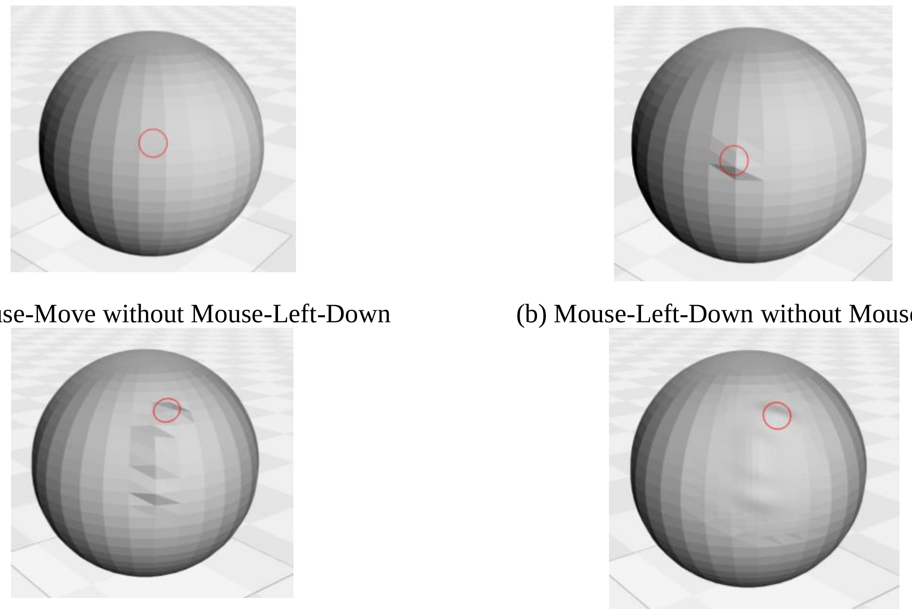
4.2 Implementation Flow

The very first stage to start sculpting is to enable the sculpt toggle, triggering the constructions for octree and half-edge structures. Then the sculpting worker closely listens to various mouse events. The following describes the call-back functions when different mouse events are caught under the condition that the cursor intersects with the selected model.

a) **Mouse-Move without Mouse-Left-Down**
   An intersection test is performed and if there exists an intersection point, the normal of the face intersected is computed and is used as the circle centre together with the user-defined radius of the sculpting brush to further produce the brush region (Figure 4.1(a)).

b) **Mouse-Left-Down without Mouse-Move**
   This even can be regarded as the positive signal that users decide to start doing sculpting from the current mouse position and it acts as a setup stage for following sculpting calls (Figure 4.1(b)). According to the effect predefined by the users, it may keep a record on some fixed arguments which function as references for the force computation for sequential calls. For instance, under the dragging effect, the intersection point's position is saved on this event and is regarded as the starting position of the mouse moving trajectory while the ending position is the intersection point' position in a subsequent mouse event, using the information of which the moving direction and distance can be computed to determine how

the dragging effect will change the selected vertices. Other information such as vertices inside the brush sphere, the normal vectors of these vertices can also be computed and be stored as “previous-state” data.


(a) Mouse-Move without Mouse-Left-Down (b) Mouse-Left-Down without Mouse-Move

(c) Mouse-Left-Down with Mouse-Move (d) Mouse-Left-Up

Figure 4.1 Effects of mouse events when sculpting.

c) Mouse-Left-Down with Mouse-Move

This is the core function where the sculpting effect truly takes place by determining who and how the effect should be applied, and in fact applying the effect, as well as rendering the change in real-time on the screen (Figure 4.1(c)). “Previous-state” and “current-state” information is implicitly maintained and are transited on this event.

Similar to the event above, different arguments are recorded for different sculpting effects so as to support subsequent function call. Additional to these “instinct” data which remains the same as long as the vertices’ information remains the same, changing data is also calculated for evaluating how the detail changing mechanism of a sculpting effect can be applied. As the dragging effect mentioned above, trajectory direction change is computed on this event and is used to immediately modify the 3D position of the selected vertices so that a real-time sculpting result is produced and is rendered on the screen.

d) Mouse-Left-Up

This event indicates an end of a sculpting stroke, and it clears sculpting states' information containers so that the starting point of a next stroke will not be affected by the previous stroke. This is also a stage for performing optimization to the stroke newly created such as smoothing thus the quality of sculpting can be improved (Figure 4.1(d)).

5 Result Discussion and Suggestion

5.1 Evaluation of Skinning Surface

Generally speaking, creating a side surface through skinning technique lends the program the power to deal with free-formed surface (Figure 5.1). The surface generated can flow smoothly and form a reasonably good result which can, to some degree, be predicted by users according to the B-Spline shapes on the pages of the canvas. By using control points to construct the cross-sections, users are not required to manually add a large number of points to fix the curve compared with the use of interpolation points. The displacement vector further reduces the workload of transiting a cross-section to a new 3D position compared with the work to produce the same result required by manually moving all the points on the canvas page to a new 2D position.

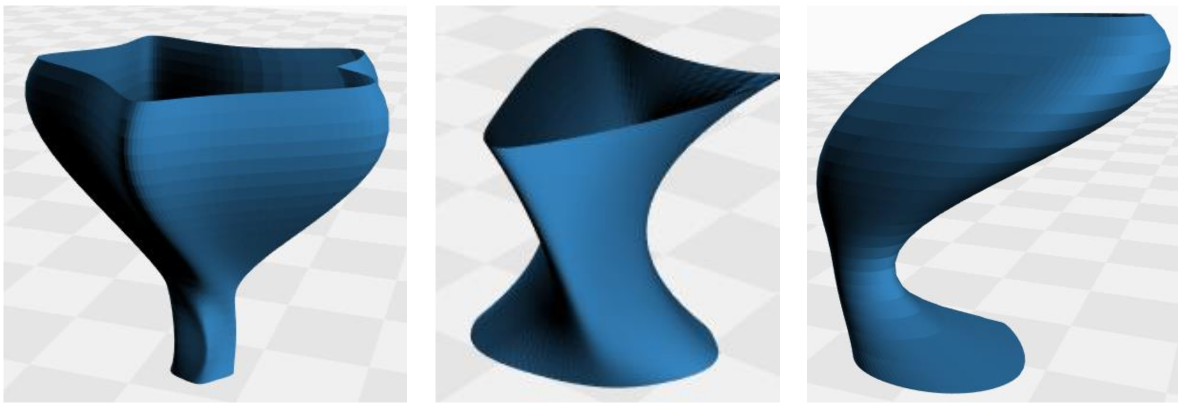


Figure 5.1 Applications of skinning surface.

However, since viewing the model uses the camera of the scene, a user need to repeatedly adjust the camera position to get a full picture of the skinning model. This can cause the user to loss the perception of the x and y directions due to frequently rotating the camera. Recalling that x and y directions are changeable parameters when modeling the surface, it may result in confusion and cause the user to take unnecessary operations to test the correct transition direction of x and y under the current camera position.

This problem may be solved by providing a secondary camera which uses the center axis of the targeting model as the axis for its rotation center and has a fixed orbit set according to the 3D bounding box of the model. Therefore, by rotating the secondary camera, the eye level remains the same and the result of different viewing angle can be produced in a secondary viewport, avoiding leading to the change of the world camera coordinate system and the confusion in x and y directions while still providing the user with a full perception of the model.

5.2 Evaluation of Digital Sculpting

Although the implementation is able to apply sculpting effect on the model surface and present the result back to the user in real-time with acceptable speed and memory consumption with the complementary data structures of octree and half-edge, the result itself falls to meet a high satisfaction level. The force applied is oversensitive to both the mouse move event and the mesh density and thus the result produced is unstable and hard to predict. Different moving rates of the brush can produce very different sculpting results. As shown in Figure 5.2, the differences among the moving rates of three strokes are reflected in the sharpness of the generated bumps. It may be utilized to control the smoothness of the sculpting effect. However, generally it should not be the case since the moving rate of the cursor depends on excessive factors and is hard to be controlled by an untrained person which is actually the target audience of the project. Besides, different levels of smoothness can be achieved by repeated sculpting rather than relying on the subconscious moving speed.

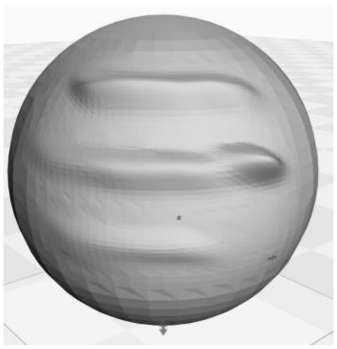


Figure 5.2 The sculpting results produced by different moving speeds of the mouse. The speed increases from upper to lower results.

The flow of how a sculpting effect should be applied and the design of algorithms of how the force should be calculated remain a significant part which can greatly contributes to the quality of the outcome of sculpting. Since currently, the sculpting effect is applied in real-time, when the updating direction of a vertex results in the vertex growing to obstruct the moving path of the cursor, that vertex can experience unexpected dramatical transformation since the sculpt effect which should be applied to the vertices behind it is kept being applied onto the blocking vertex itself (Figure 5.3). A swap in the execution orders of applying the update to current active vertices and detecting next set of vertices to be transformed can release the problem to some extent, but not outstandingly.

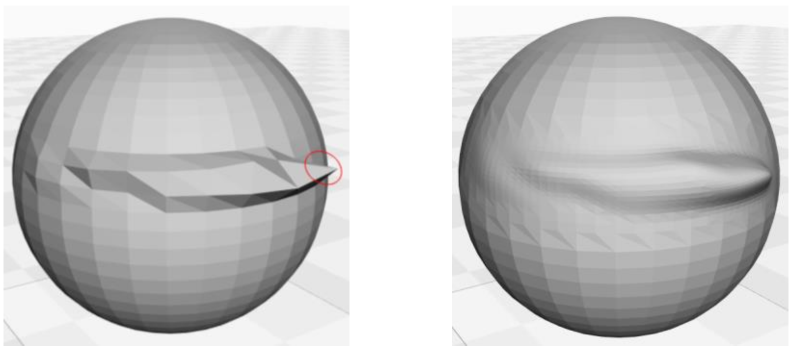


Figure 5.3 A vertex growing to block the moving path of the brush experiences undesirable transformation.

Moreover, due to the different density levels of the meshes all over the surface resulted from incremental subdivision, the sculpting effect generates unnatural appearance which cannot fit into the original object surface (Figure 5.4). This raises the question on when and to what extent should the subdivision algorithm be executed.

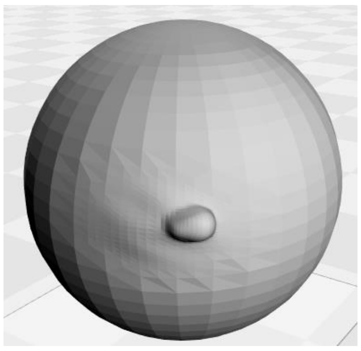


Figure 5.4 Unnatural result produced on the surface with high mesh density.

How to fit digital sculpting into a block-based modelling style platform remains the main topic to be explored in the future. The algorithms of sculpt effect shall be further researched so that the effect of each iteration can be limited within a controllable range and thus sculpting can be performed with high error tolerance. As a counterpart to subdivision, decimation can be the potential solution to the surface appearance problem caused by various mesh densities while providing a further optimization on memory usage [17].

6 Conclusion

This final year project, an easy 3D modelling tool, explores the possibility of adding detailed surface deformation functions to the block-based modelling style. The two proposed options are raised from different aspect, with the skinning surface modelling based on B-Spline curving functions and the digital sculpting based on modification of basic units of vertices and polygons. Throughout the implementation, the design of the two options are guided by the criteria of being user-friendly and generating high-quality models, which requires the optimization methods of computational speed and memory consumption being taken into careful consideration and as a result, multiple data structures are adopted to fully utilize their advantages on different function designs. Octree is used for intersection test and half-edge is used for subdivision. Notably, since subdivision and sculpting modify the vertices' positions, updates in these two data structures are frequently occurs and algorithms are designed to not recursively cause rebuilds.

At current stage, both options have been implemented in this project. While the skinning technique is capable of generating free-formed surface, digital sculpting remains to be fail behind expectation due to the problems existing in algorithm's design. Further research is suggested to explore the function algorithms for not only the sculpting effects but also the execution sequence of picking active candidates and applying effect. Other auxiliaries can also be added to release the users' workload and upgrade the program performance, such as a secondary camera and a corresponding viewport to support skinning surface adjustment and surface decimation.

The anticipation behind the project is to bridge the gap between the hard-to-acquired 3D modeling skills and easy-to-obtain 3D printing hardware and software, and thus assist in the spread of 3D printing technologies to be far-reaching. Consequentially, the boom of the personal and customized 3D printing business, ranging from manufacturing industries to academic development can be expected. This project can also be regarded as an experiment on what kinds of 3D modeling options can be integrated into a platform adopting a block-based style, under the design guideline that primary users can get started easily while the modeling options are robust and timely.

References

[1] S. Mishra, "3D Printing Technology," Science Horizon, vol. 10, pp. 43-45, October 2014.

[2] B. C. Gross, J. L. Erkal, S. Y. Lockwood, C. Chen, and D. M. Spence, “Evaluation of 3D Printing and Its Potential Impact on Biotechnology and the Chemical Sciences,” American Chemical Society, Washington, D.C., Rep. 86-(3240−3253), 2014.

[3] T. Rayna, L. Striukova, J. Darlington, “Co-creation and user innovation: The role of online 3D printing platforms,” Journal of Engineering and Technology, vol. 37, pp. 90-102, July–September 2015.

[4] 3D Builder Online Software. (n.d.). SketchUp [Online]. Available: https://www.sketchup.com/products/sketchup-free [Accessed: 2017, Sep 2]

[5] Sketch-Based Modeling. (n.d.) Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Sketch-based_modeling [Accessed: 2017, Dec 28]

[6] VECTARY - the free, online 3D modeling software. (n.d.). VECTARY [Online]. Available: https://www.vectary.com/ [Accessed: 2017, Sep 2]

[7] Tinkercad | Create 3D digital designs with online CAD. (n.d.). Autodesk [Online]. Available: https://www.Tinkercad.com/ [Accessed: 2017, Sep 2]

[8] L. A. Piegl, W. Tiller, "Surface skinning revisited," The Visual Computer, vol. 16, pp. 273-283, June 2002.

[9] C. -K. Shene, "CS3621 Introduction to Computing with Geometry Notes," Michigan Technological University, May. 2011. [Online]. Available: http://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/notes.html [Accessed: 2017, Sep 10]

[10] "SMOOTH". (n.d.). Pxiology. [Online]. Available: http://docs.pixologic.com/user-guide/3d-modeling/sculpting/sculpting-brushes/smooth/ [Accessed: 2017, Dec 17]

[11] K. Matsuda, R. Lea, WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL, 1rd ed. Addison-Wesley, 2013.

[12] E. Nevala, "Introduction to Octrees," gamedev.net, Jan. 20, 2014. [Online]. Available: https://www.gamedev.net/articles/programming/general-and-gameplay-programming/introduction-to-octrees-r3529/ [Accessed: 2018, Jan 23]

[13] J. Havel, A. Herout, "Yet Faster Ray-Triangle Intersection," IEEE Transactions on Visualization and Computer Graphics, vol. 16, pp. 434-438, July. 07, 2009.

[14] R. F. Hernandez, "Dynamic Subdivision Sculpting," Hermanos Saz University.

[15] H.-R. Pakdel, F. F. Samavati, "Incremental Subdivision for Triangle Meshes," International Journal of Computational Science and Engineering, vol. 3, 2017.

[16] M. McGuire, "The Half-Edge Data Structure," flipcode.com, Aug. 07, 2000. [Online]. Available: http://www.flipcode.com/archives/The_Half-Edge_Data_Structure.shtml [Accessed: 2018, Feb 8]

[17] W. J. Schroeder, J. A. Zarge, W. E. Lorensen, "Decimation of triangle meshes," SIGGRAPH, 1992.