Speedy Sketching

A Dissertation Submitted to The University of Manchester

For The Degree of Master of Science

In The Faculty of Engineering And Physical Sciences

2009

Kenneth Pak Kiu Lam

School of Computer Science

Contents

List of Figures

Abstract

Speedy Sketching by Kenneth Pak Kiu Lam

Supervised by Toby Howard

Nowadays, a lot of applications exist for the purpose of constructing 3D models. However, many of them put too much emphasis on detail and neglect usability and the speed of construction. Speedy Sketching aims at providing a very simple interface for sketching simple shapes quickly, preferably with a pen or mouse interface, which will provide freedom and naturalness for the users. The lack of details will encourage quick iterative adjustments, and designers will focus more on the overall structure.

The idea behind Speedy Sketching is the structure of a lot of complex shapes can be composed of simple shapes. 2D and silhouettes of simple 3D shapes sketched by the user will be automatically analysed and generalised, removing human errors in the process. It is hoped that this innovation will greatly increase the speed and intuitiveness of 3D model construction.

Declaration

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

i. Copyright in text of this dissertation rests with the author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author. Details may be obtained from the appropriate Graduate Office. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the author.

ii. The ownership of any intellectual property rights which may be described in this dissertation is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

iii. Further information on the conditions under which disclosures and exploitation may take place is available from the Head of the School of Computer Science.

Acknowledgements

I would like to thank my supervisor Toby Howard for his motivation and advice. There are times when I feel frustrated but his encouragements will always put me back on the right track.

Although my family and I are in different parts of the world, I can always feel their unconditional love.

I am grateful for my company as they allow me to take time off of job to complete my Masters dissertation, a critical and final stage of my academic life.

Last but not least, hats off to the following people for participating in the user survey (in the order of submitting the survey): Thomas Sharp, Peter Mcnerney, Gaetan Deputier, Samantha Bail, Viridis Liew, Eddy Seager, Alvin Liang, Alan Lam, Eric Chan, Abdelhak Attou, Lauriane Lancereau and Desmond Choy. This dissertation will not have been complete without you guys.

– Kenneth “KeniF” Lam

The Author

The author has graduated in (BSc) Computer Engineering in the University of Manchester. His undergraduate dissertation *SodaSumo* (2008) involves the extension of *SodaConstructor*, a simple physics platform for the simulation of spring-mass models. By introducing collision detection between two models, a new form of competition – sumo wrestling – for artificial life spawned. *SodaSumo* has recently been cited by Grupo de Estudios en Biomimética (GEB/Research Group in Biomimetics) at the University of Malaga, Spain.

SodaSumo was released to the Sodaplay community and has been used by over 100 people around the world.

1 Introduction

1.1 The Need for “Casual” Sketching Tools

Living in the age of technology, it may seem surprising that the first pictures we draw in our childhood are with a pencil and a piece of paper. Sketching is the most fundamental way of image representation, and humans are automatically adapted to it. Although it could mean that lines are not straight and shapes are not closed, it never stops us from expressing our ideas through sketching them quickly in diagrams which, often, only the creator can understand.

The ambiguity of a sketched object amplifies if it is supposedly 3D. A sphere becomes a circle and a cube is flattened, and there is no depth associated with anything. It is impossible to rotate a flat sketch which makes presentation difficult. A lot of professional software exists for the purpose of generating 3D models. However, the key word here is professional – there are too many operations, a number of grids are often included for accuracy and most importantly, there is no way to sketch a desired object naturally. These things intimidate casual users and more often than not, they stop using the software before being able to draw anything meaningful. Clearly, there is a need for 3D modelling software specially designed for casual users.

Speedy Sketching aims at filling a void in the 3D model design industry where most of the software is difficult to learn. Although this scenario is starting to change in recent years with the likes of Spore – which allows creating 3D creatures and sharing them online – spawning in the market, it does not solve the fundamental problem of not allowing users to sketch naturally, just like when they

are given a pencil and a piece of paper. With Speedy Sketching, the operations should be easy to learn. Being able to sketch objects in a WYSIWYG (What you see is what you get) manner, creating 3D models will no longer just be done by professionals – it can be a pastime for casual computer users as well. 3D model creation is essentially an art, and everyone should be given the same chance of expressing their ideas, just like what we have with 2D programs.

1.2 The Research

The main priority of Speedy Sketching is to research new methods of sketching 3D models. There is a lot of existing innovative ways to create 3D models quickly, like the extrusion operation provided in Google SketchUp where any closed 2D shape can be extruded to a prism (Figure 2.7). However, the author believes that more can be done to simplify the process. Less effort will be made to replicate already proven methods of sketching, because only innovations will drive a field of research forward.

This report starts with brief background information on sketch interfaces (Chapter 2), explaining its relationship with creativity and gives an insight on some popular sketching software. From existing applications, we can learn the advantages and disadvantages of common implementations and then decide what to replicate or improve. Chapter 3 presents the concepts which act as the backbone of this research and covers the design stages.

In Chapter 4, details of implementation and various shape detection algorithms developed by the author will be explained thoroughly, together with the reasoning for changing some of the implementation methods. This is followed by the results (Chapter 5) of the developed application which will present the outcome of the designed algorithms and interfaces. It will include the limitations of the

application for potential expansions or editions to this research.

As Speed Sketching is highly user-oriented, a user survey has been carried out for obtaining feedback. The results and observations will be detailed in Chapter 6. A conclusion will then be provided in Chapter 7, citing the main findings of this research.

1.3 Assumptions

It is assumed that the readers of this thesis have a basic understanding of trigonometry and 2D and 3D geometry.

2 Background

2.1 Sketch Interfaces

2.1.1 Introduction

Traditionally, computers have been more machine-oriented and made to worry less about the convenience for users. But as computing power improves two-folds every 2 years [2], computer scientists have more in stock for boosting user experience. Nowadays, the computer can process properties like ambiguity, creativity and informal communications, thereby allowing support for user-oriented tasks like drawing, speech and design [1]. This gives rise to sketch interfaces which allows users to scribble ideas at will with minimum constraints.

2.1.2 Sketching and Creativity

With minimum constraints comes great innovations. It is shown that sketching and gesturing are 2 modes of informal and perceptual interaction which are valuable for creative design tasks [3]. It is argued that the ability to rapidly sketch objects with uncertain types, sizes, shapes and positions is important to the creative process. The ambiguity encourages designers to explore different ideas without being burdened by colours, fonts, precise alignment and formality. In a study performed by V. Goel [4], designers were asked to solve design problems either by sketching on paper or using traditional computer design applications. It is found that designers quickly created variations of an idea in free-hand sketch. In contrary, designers who used a computer-based drawing program spent more time fiddling with the initial design, thus limiting creativity.

The above study does not imply computers should be avoided for creative

design work. Instead, we should combine the freedom provided by a sheet of paper and the efficiency of computers to create powerful applications – *electronic sketching* systems. The strength of such systems lies in the ability to recognise graphical elements common to a particular domain when they are drawn [1]. For example, in MathPad^2, mathematical equations, numbers, functions, trigonometry scribbles and even simple diagrams can be recognised (Figure 2.1) [5]. With a paper-like sketch interface, users are more encouraged to brainstorm. It does not matter if a mistake has been made, because a gesture is provided to easily rub out sketches [6]. The fault-friendly interface enhances scientific discussions where different ideas are expressed.

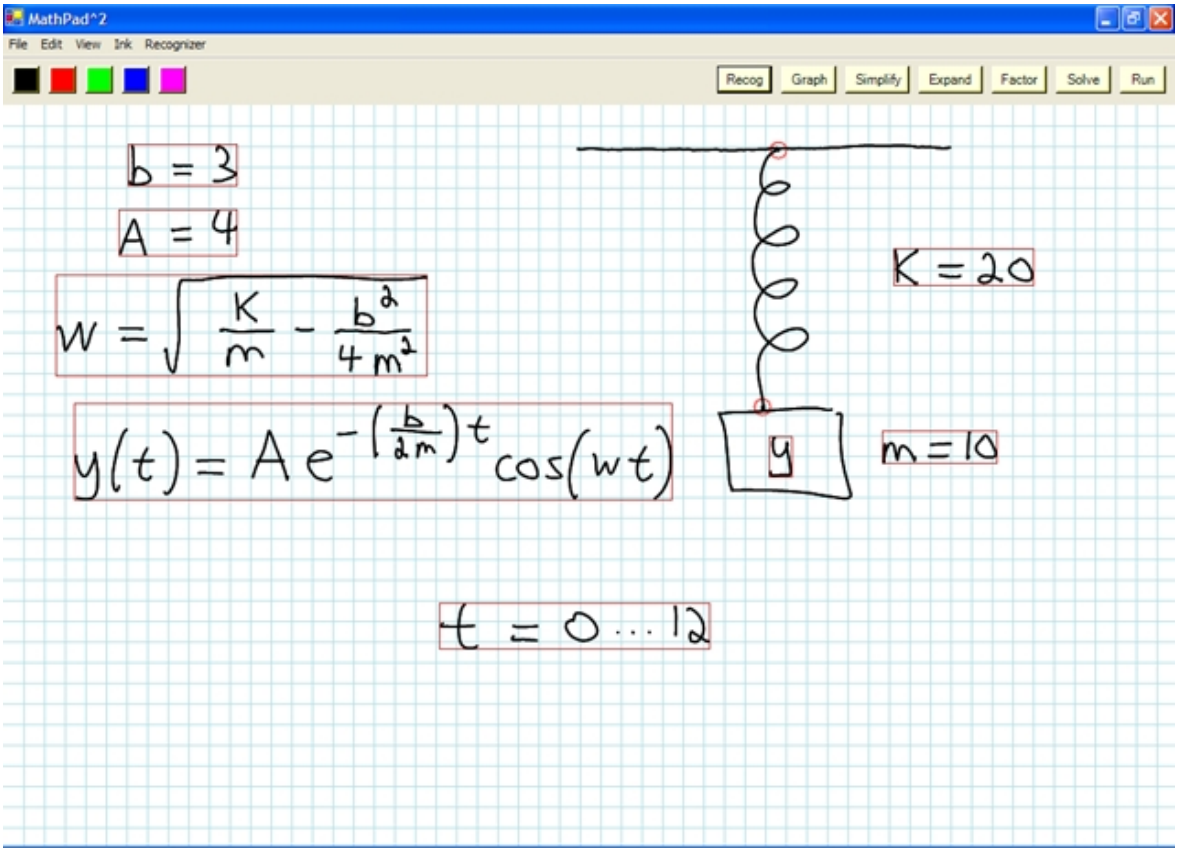


Figure 2.1: Mathematical equations, symbols and diagrams can be recognised in MathPad^2 (image from [5])

2.2 Interface Design

Interface design is crucial to an application's success but often overlooked by developers. It is even more important for sketch interfaces because users are usually given more freedom than in traditional software - the way information is input affects the outcome to a greater extent.

Usually, good interfaces have the following characteristics: easy to learn, effective to use, and provides an enjoyable user experience. They make use of familiar physical objects, are aesthetically pleasing and have one-step actions to perform core tasks. Sharp *et al* (2007) give an example of a good interface, the TiVo remote. On the remote, the icons are large, clearly labelled and logically arranged. The cartoon icons and colourful buttons also make every button distinctive and a little time is needed to tell what it does.

Figure 2.2: The TiVo remote, TiVo Inc,. (image from [28])

The key to making a successful computer interface is no different. A designer needs to know the kind of possible activities performed by the user, taking into account what people are good and bad at. Also, they need to consider what might help people with the way they currently do things and how to improve user experience. Last but not least, is to get feedback from potential users in the form

of questionnaires and interviews [27]. With a good interface, users are more likely to stick to a product; with a bad interface, users will be too intimidated to even get started.

2.3 Cartographic Generalization

2.3.1 Introduction

Cartography – the making of maps – has close relationship with sketch interfaces. It is because they both deal with arbitrary input in the form of points, lines and shapes. Digitalisation of maps allows applications to display them at different scales, sizes and levels of detail according to requirements. However, it would make sense to display the map at a lower resolution only if the amount of data processed is reduced correspondingly. Otherwise, computing time would not be significantly improved. Traditionally, map generalization is a tedious task for cartographers, who have now turned to technology to automate the process. With the help of computers, map generalization can be achieved more uniformly, precisely, rapidly and with lower cost [7].

2.3.2 Line Simplification and Smoothing

A number of generalization techniques are used by cartographers, but only 2 of them are of interest to Speedy Sketching – line simplification and smoothing. Line simplification refers to a reduction of density by selecting a subset of the original points, retaining points most representative of a line [8], which will reduce computing time and storage. Smoothing, on the other hand, relocates or shifts coordinate pairs in an attempt to plane away small perturbations and capture the most significant trends of a line, which will result in a more aesthetically pleasing representation [9]. These two techniques are useful for reducing the human errors introduced when using a pen interface and increasing the speed of calculations. Figure 2.3 below illustrates line simplification and smoothing operations.

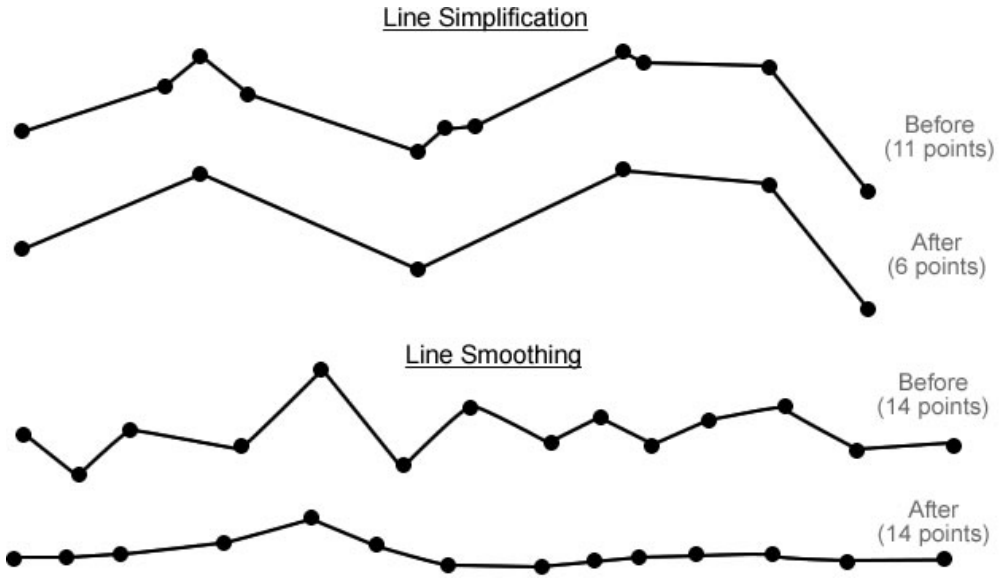


Figure 2.3: Demonstration of line simplification and smoothing. Note the change in number of points.

2.4 Previous Work

2.4.1 Teddy/SmoothTeddy

Teddy is a Java application by Takeo Igarashi (1999) which allows drawing of rotund 3D models in a sketch-based environment. Despite the very simple user interface, Teddy detects the user's free-form strokes and presents a number of complex operations like extrusion, cutting, digging, line painting and erasing of lines. Initially, the engine takes a circular 2D shape and predicts the thickness of the object. In general, wide areas become flat and narrow areas become thin [10]. Figure 2.4 demonstrates a number of operations in Teddy.

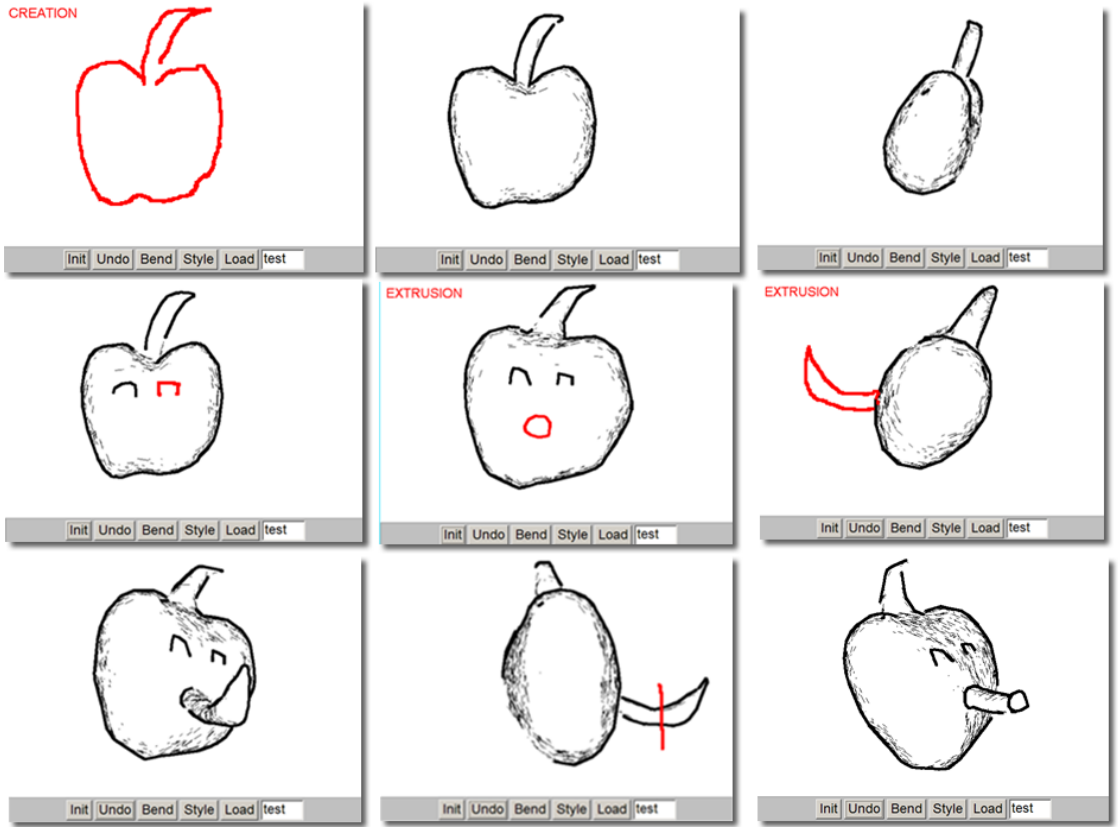


Figure 2.4: Basic operations of Teddy

In 2003, an update to Teddy – SmoothTeddy – was released. Apart from the original features of Teddy, it now generates a hierarchical structure of the object automatically, which can be exported in VMRL format and used in animation software like Alice [11]. A main improvement is that polygon meshes which are originally rough and with uneven triangulations are refined to increase the smoothness [12]. Figure 2.5 below shows a screenshot of SmoothTeddy. As shown, the prediction algorithm is quite robust for natural objects like rocks and cartoon animals. Users have no need to worry about the depth of the objects, and can paint directly on a 3D surface. The interface is also extremely easy to learn as gestures like cutting are available.

However, SmoothTeddy is only limited to the drawing of naturally rotund objects. “Artificial” objects like cylinders, cones and rectangular prisms cannot be drawn. Also, all objects must be connected. There is no option to create 2

separate objects or an object with a floating component. The application was developed 6 years ago but error messages are seen almost every time it is run, adding to the inconvenience. Finally, as shown in Figure 2.4, the body of the "apple" appears to be flat. It is because the algorithm assumes the depth of the object automatically and it cannot be explicitly defined by the user. All in all, SmoothTeddy does its intended job well, which is to draw puppet-like round creatures.

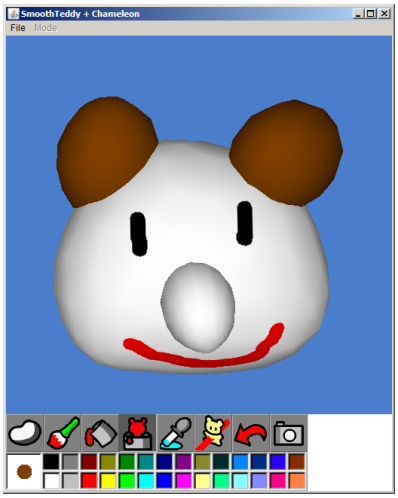


Figure 2.5: SmoothTeddy features much smoother models

2.4.2 3D Journal

3D Journal is an interactive tool for freehand sketching developed by Cornell University (2005). It aims at combining the ease and speed of freehand sketching with the flexibility and analytical abilities of Computer-aided design (CAD) tools. To sketch a 3D model, the user will draw 2D sketches of straight and curved strokes, which will contain all edges as if the object is transparent. An algorithm will then determine the angular distribution of the strokes and offer a z-value to each vertex to create an orthogonal 3D axis system [13]. New strokes can be added onto constructed objects, which increases the flexibility of the system. This flexibility allows the initial sketch, reconstruction and adding detail to be done in a consistent interface, which improves efficiency. Moreover, an eraser tool can

erase edges, a cut tool can remove parts of a face, and the stiffness of the material can be changed.

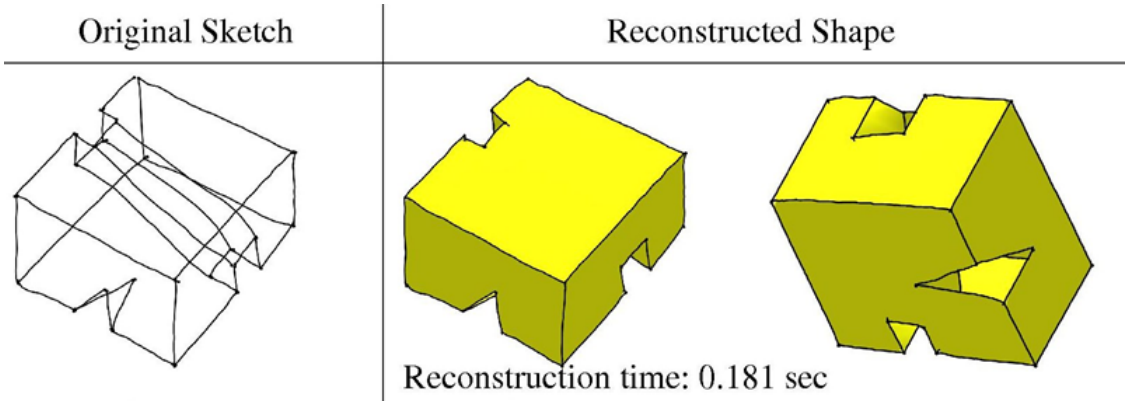


Figure 2.6: Constructing a 3D shape from 2D edges, 3D Journal (image from [14])

The system is extremely robust as small objects with about 20 strokes take only 0.2 second to construct. It combines sketching and analysis which benefits the efficiency of a CAD process. The average angular distribution of edges is used to determine the orientation of the object, which is generally accurate. However, the system can only detect shapes with perpendicular strokes and cannot be used for drawing spheres and cones. Curves may be misinterpreted as one 2D view may represent 2 different orientations in space [14]. Moreover, complex objects can contain lots of intersecting and overlapping edges that may confuse the user. Finally, for a small object with limited number of strokes, the output of the object may be affected by human errors, as the system does not attempt to generalise the models drawn.

2.4.3 Google SketchUp

Google SketchUp is a powerful 3D modelling software which can be used for architecture and design, engineering, construction and digital entertainment. It is the pioneer of the patented push/pull technology (Figure 2.7), which will be incorporated into Speedy Sketching. To create a circle, you simply need to place the centre and drag the length of the radius. The snapping tool is also extremely

powerful as it makes suggestions to snap lines to the nearest plane. It allows sharing of models with users worldwide and can be linked to Google Earth. According to Mortenson Construction, cost and speed of construction of 3D models have been vastly reduced [15] because it does not require experienced CAD designers to operate.

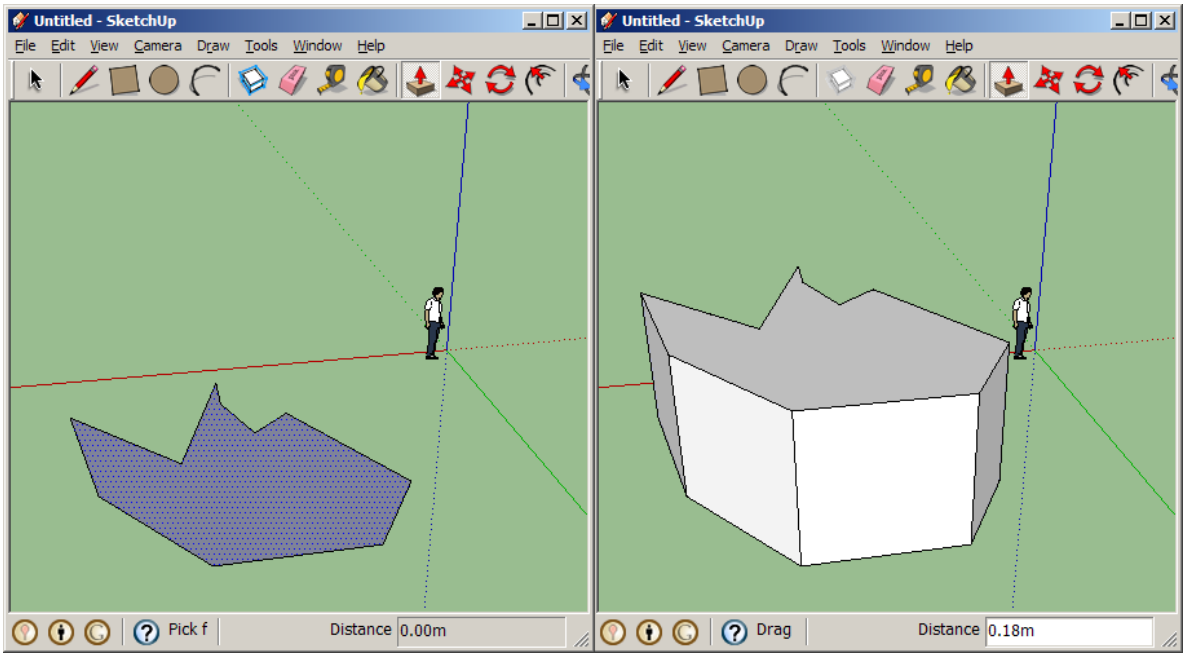


Figure 2.7: Pulling of an arbitrary polygon, Google SketchUp

However, it is difficult to draw polygons on an arbitrary plane. As shown in Figure 2.8, a seemingly planar polygon is snapped wrongly to the main planes (x/y/z). Also, drawing of cones and spheres requires the rotation of a 2D object or downloading of existing models from the web, which are slow. Finally, freehand drawing requires clicking the mouse button to create points. This reduces redundant points, but it is unnatural and users do not get the feeling that they are drawing with pen and paper.

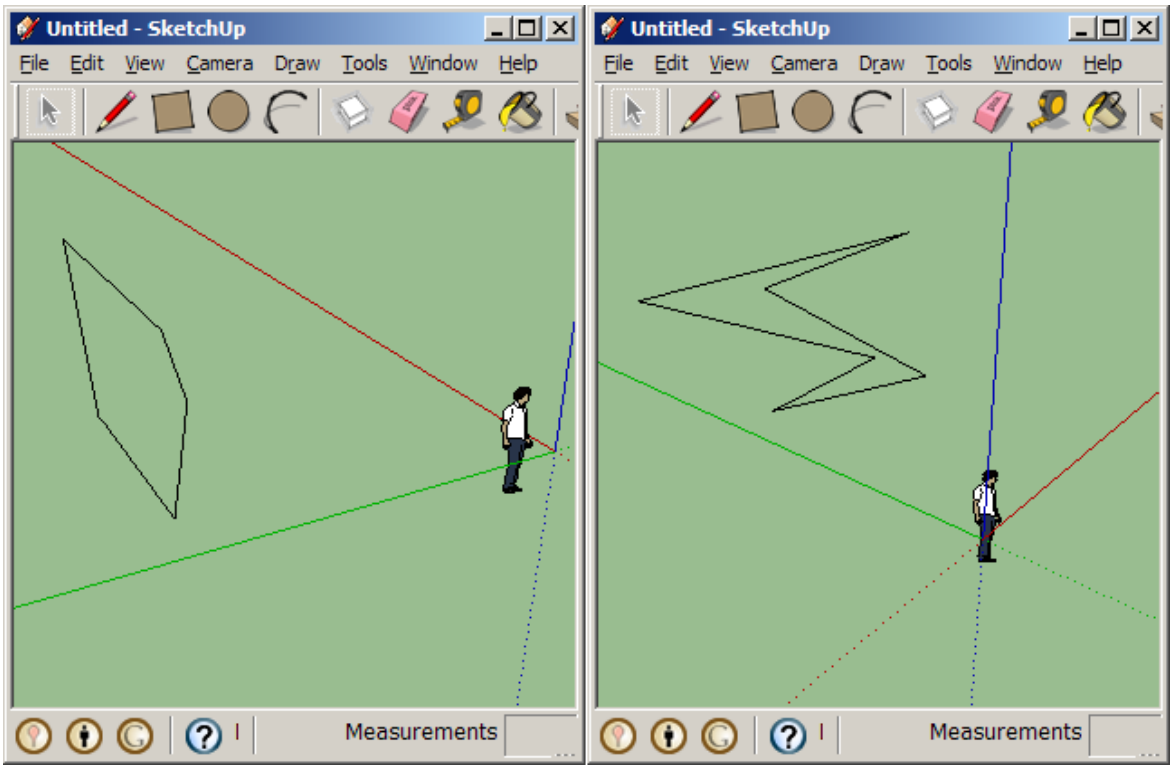


Figure 2.8: A supposedly planar polygon (left) and the scene rotated (right)

Speedy Sketching attempts to take the advantages of the above applications for simplicity and usability reasons. At the same time, we will try our best to avoid the disadvantages mentioned. Regardless, we intend to incline towards the approaches of Teddy and 3D Journal in which users sketch without many constraints or having to worry about accuracy. The design principles will be explained in the next chapter.

3 Design

3.1 Introduction

What makes a perfect drawing application? In the author's opinion, it is one that gives the illusion that you are drawing freely with a pen and a piece of paper. When we were small, we used to draw everything based on its front view, which is the easiest way to represent an object. With these two ideas in mind, Speedy Sketching will allow users to sketch objects in their front view, then convert them to 3D shapes by dragging. By clicking specific icons, users will also be able to convert front views of common 3D shapes to a 3D shape (Figure 3.5). By combining simple 3D shapes, the user can obtain the desired complex 3D object.

The interface will be designed with a strong focus on simplicity – most users should be able to understand the software fully within 5 minutes of use. The number of key strokes required for an operation will be reduced to a minimum. It is hoped that with a simple interface, users will spend most of their time drawing but not switching between views and modes.

3.2 Target Audience

Speedy Sketching is directed to any discipline which needs quick brainstorming and sketching of ideas in 3D models. E.g., Computer animators who need to make 3D storyboards to express their ideas. It can also be used in art classes in school curriculum. It should be noted that Speedy Sketching is not intended for extremely detailed final models. Instead, we focus on the first step of the creative process – generating and comparing variations of design models quickly. These models are crude, their alignment is not perfect but they stimulate thinking and

discussions.

3.3 Specification

3.3.1 Interface Basics

The interface will be so simple a user of any age can pick it up in a fairly short amount of time. The application will be operational with just a mouse or pen interface, but more focus will be put on building a pen interface which allows tapping of icons. Many drawing applications assume users to be using a mouse and bind zoom in and out functions to the mouse wheel. In Speedy Sketching, this is implemented as buttons on the screen, with the operation automatically recalled if the user holds down the buttons.

Keyboard shortcuts will be available for advanced users but not as a requirement for operation, because a keyboard is not always as readily available in a Tablet PC. All possible operations will be shown as buttons in the GUI and achievable with one click. The icons will be relatively bigger than those found in a common application, because users will often find themselves carrying a Tablet PC and sketching in an unstable environment. Also, this will make the application more usable for children.

Most computer software includes standard menu bars as well as icons for disabled users. However, it is safe to assume that a disabled user will not use a drawing application. Therefore, we have chosen to display everything as graphics for easy identification. A single click will also be quicker than going through a list of menu items.

The application will have a main window split in 2 views, one for showing the main scene (View Panel) and the other for drawing objects (Drawing Panel), which will be introduced in the next section. As rotation is a very common operation,

vertical and horizontal sliders will span the entire screen for more precise controls.

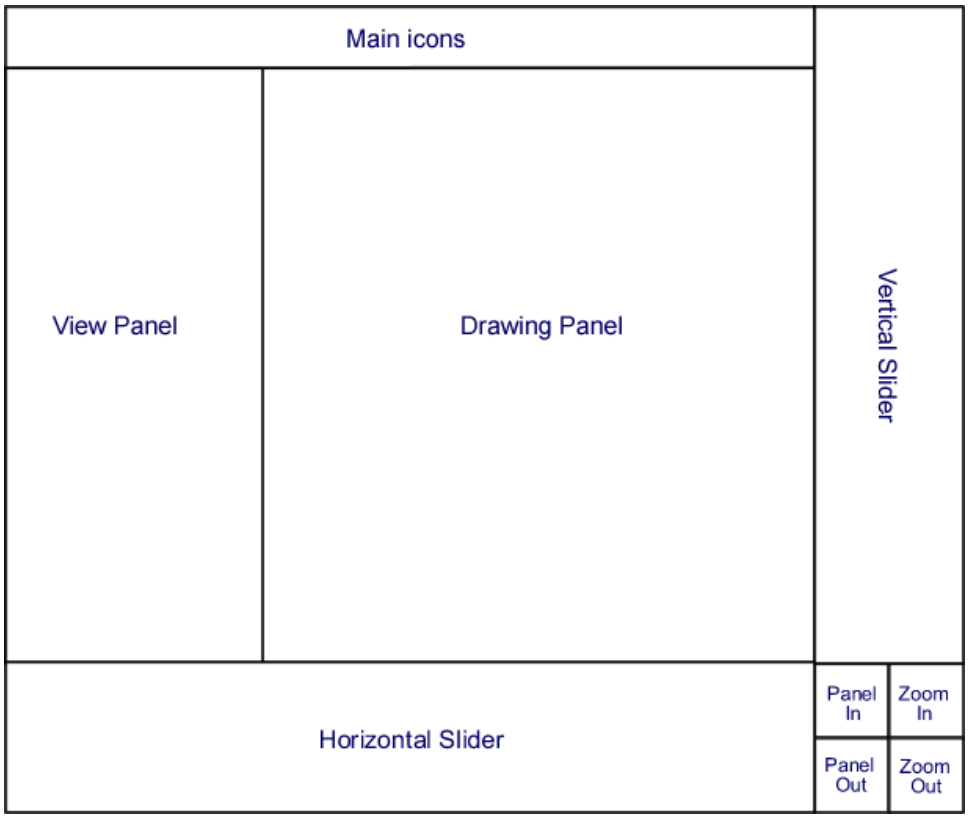


Figure 3.1: Design of the main window

3.3.2 View Panel

The View Panel shows a perspective view of the scene. It displays far objects smaller and close objects larger, giving a more realistic view of the scene. As it is not the main drawing area, it dynamically covers one-third of the window, giving a supplementary view. The Drawing Panel is not visible as the user will not be able to draw here, showing a clearer view of the scene.

3.3.3 Drawing Panel

The Drawing Panel will handle all object editing and sketching in Speedy Sketching. It displays the scene in the orthogonal view, and is always parallel to the screen. With the left mouse button (or the equivalent), users can draw shapes consisting of one or more line strokes. The Drawing Panel can be slid in the z-

axis to cover all space (Figure 3.2). It will be semi-transparent, allowing the user to see the objects behind, while showing the bisection caused by it. The bisection allows users to tell where in space they are drawing.

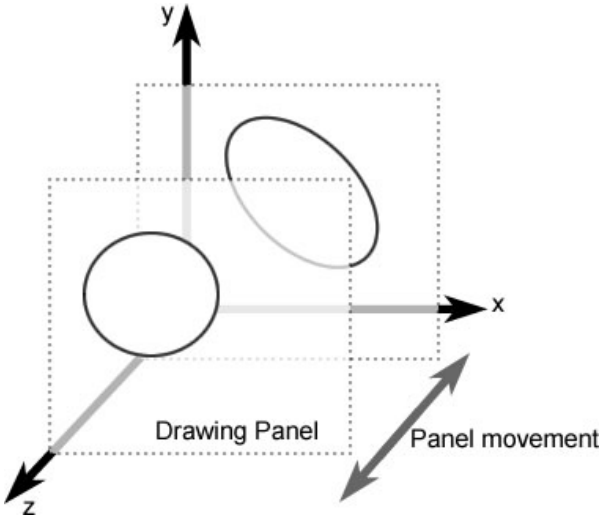


Figure 3.2: Movement of Drawing Panel in 3D space

3.3.4 Drawing

Speedy Sketching will feature two 2D to 3D conversion modes. Firstly, any *closed* 2D shape can be extruded to form a 3D prism (Figure 3.3), which will be centred at the Drawing Panel. Secondly, the user can select the basic 3D shape they want, and draw the front views in 2D (Figure 3.4). Speedy Sketching will automatically determine the parameters (orientation, size, width, height, radii) of the 3D shape, increasing the sketching speed and also providing a way to fix human errors. The resulting 3D shapes will also be centred at the drawing panel. A lot of objects in our daily lives show reflection symmetry and are perfectly symmetric on both sides of a mirror line. This property can be exploited in Speedy Sketching.

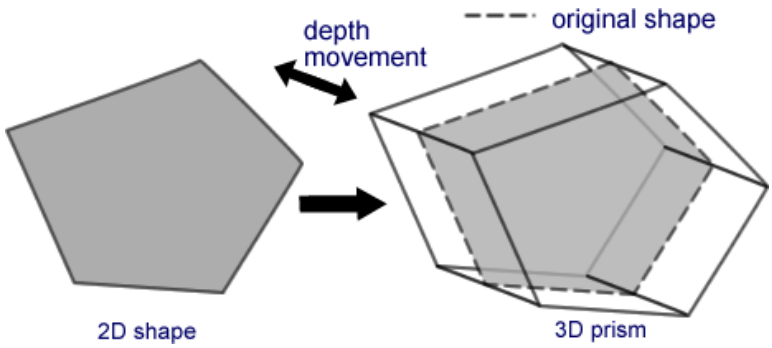


Figure 3.3: Extruding a 2D shape into a 3D prism

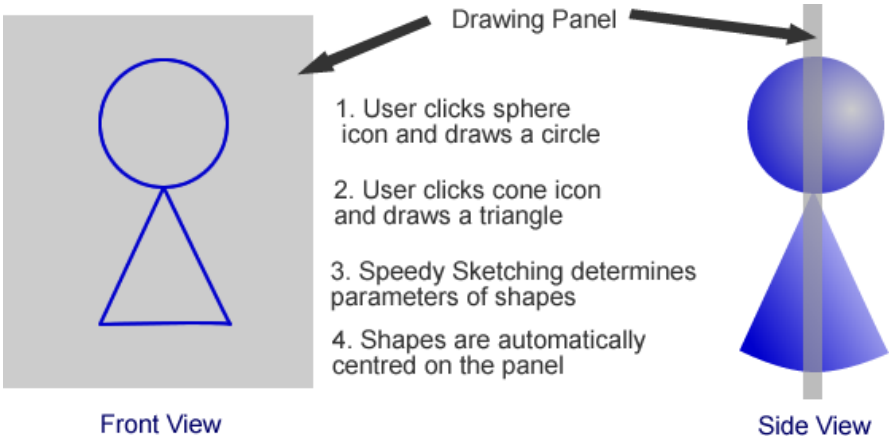


Figure 3.4: 2D silhouette to 3D conversion

Finally, by switching to delete mode, individual objects can be deleted. As the user moves the cursor around the scene, certain objects will be highlighted to show the object to be deleted.

3.3.5 Shape Detection

Depending on the shape, user behaviour and shape orientation, users may draw their desired shapes with a varying number of strokes, which can also vary in draw order and direction. The ultimate aim of Speedy Sketching is to determine correctly the shape parameters with a number of strokes up to the number of sides of the shape. (i.e. 1 for a circle ,3 for a triangle and 4 for a rectangle. Correctly speaking, a circle has an infinite number of sides, but most people will draw it in one stroke.) If the user decides to draw a shape with the number of

strokes less than the number of sides, (i.e. 1 or 2 strokes for a triangle), they will have to activate the detection engine by right-clicking or moving the cursor into the View Panel.

It has been suggested that a timer can be implemented for automatically activating the detection engine after a certain period of time, but different users have different drawing speeds. If the user tends to draw really quickly, then they will be waiting for a long time after each drawing. Similarly, if a user tends to draw slowly, the engine may always be activated before they finish, leading to inconvenience. Also, users could stop in the middle of drawing a shape to think about how to draw it properly. This kind of unpredictable behaviour is common during sketching and there is currently no way for the engine to determine whether a shape is complete. The addition of a timer may introduce both benefits and disadvantages which will lead to no overall improvement. Therefore, until a better solution is deduced, the idea of a timer has been dropped.

The following shape detection and conversions will be achieved:

- Circle detection / Circle to Sphere conversion
- Triangle detection / Triangle to Cone conversion
- Rectangle detection / Rectangle to Cylinder conversion (Figure 3.5)
- Freehand line simplification
- Extrusion of Rectangle and Triangle

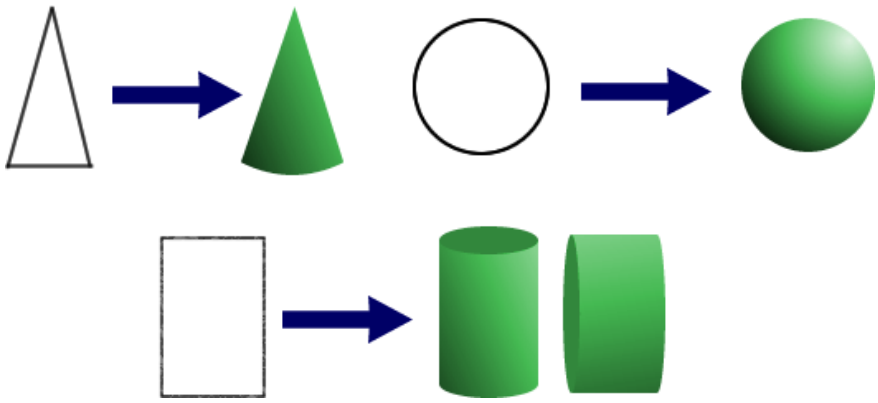


Figure 3.5: 2D silhouette to 3D conversions

3.3.6 Camera

The camera is designed such that users will be able to rotate the scene/objects in the x and y axes quickly with 2 sliders even when drawing. Unlike a virtual track ball, sliders will make sure only one axis is rotated each time. Users will then be able to rotate the scene quickly without worrying about accuracy. This is particularly useful when the user wants to make sure the scene is always upright. The sliders will allow rotation of -180 to 180 degrees and snap angles to the nearest degree. They bounce back to 0° position after each movement, so the user will not have to remember where the slider button is. The consistency of the GUI is maintained.

Unlike popular software, Speedy Sketching will *not* feature 3 views of an object under construction because it is potentially confusing and restricts the user to 3 views only. Also, we want the users to feel that they are drawing on a piece of paper.

3.3.7 Model Importing/Exporting

Speedy Sketching can save unfinished scenes in the XML format for specific information about common 2D/3D shapes. It will also be able to import and export 3D scenes and models in .obj format which is commonly used in design

software. This allows designers to modify their sketched objects in detail in after the initial creation process.

3.4 Project Plan

The project will begin with GUI design because it is the only possible way for data input. Also, to test various algorithms, the designer will be required to draw diagrams for evaluation and viewing the results. Moreover, it is best for the designer to use the GUI like a user as much as possible, so it can be improved continuously.

The next chapter explains how the design principles are implemented.

4 Implementation

4.1 Choice of Platforms

C++/OpenGL have been the choice of computer graphics and gaming industries for years. C++ is known to be extremely efficient and has a speed advantage over the increasingly popular Java. On the other hand, OpenGL is designed to be full featured and runs efficiently on a wide range of graphics architectures. The package is available for free and is compatible with Apple Macintosh, Microsoft Windows as well as Linux and embedded devices. The scalability and availability make OpenGL an industry standard for graphics applications [17]. The OpenGL API is also commonly known as the easiest graphics API for computer scientists to learn and use. For these reasons, the C++/OpenGL combination is chosen for the implementation of Speedy Sketching.

However, for a GUI-based application like Speedy Sketching, a powerful GUI toolkit is required for maximum usability. The GLUT API only provides minimal interaction support, i.e. callback methods for keyboard and mouse key presses and a simple pop-up menu. Hence, the author has turned to QT for the production of a powerful GUI. QT is a cross-platform integrated development environment (IDE) for GUI design. It provides a widget to render graphics with the OpenGL API [18] so the time required for embedding an OpenGL application in a GUI will be reduced.

4.2 Data Structure

The data structure will be object-oriented which is the common approach of modern software engineering. Since Speedy Sketching will involve common simple

shapes, the data structure can be simplified to contain parameters specific to that shape instead of all points data. Despite the reduction in space complexity, this will require Speedy Sketching to calculate the shapes dynamically which will increase the time complexity. However, all of the basic shapes are provided by OpenGL internally and highly optimized, so there will be little influence in performance. This also allows any saved data to reduce in size for faster loading. The following objects are stored and manipulated in the application:

Point: A Point contains 3 Euclidean coordinate values (x, y, z) in the floating point representation. It is manipulated by the Line object when the user sketches a line.

Line: A Line is a doubly linked list of points and can have an unlimited length (until memory runs out, which is highly unlikely). When lines are being created, a bounding rectangle will be actively maintained by storing the maximum and minimum coordinates of the x and y axes. This is required by the circle detection algorithm, and may be required by other algorithms if the program is expanded.

Strokes: A Strokes object encapsulates all user strokes for a single object, and is passed to the various detection algorithms for identifying parameters of a shape.

Shape: Contains information about rotations in the x and y axes which are commonly found in all shapes. Circle, Rectangle, Triangle, Cone and Cylinder all extend this class. Each one of them encapsulates shape-dependent parameters like radius, centre, base length and width.

A simplified class diagram is shown in Figure 4.1.

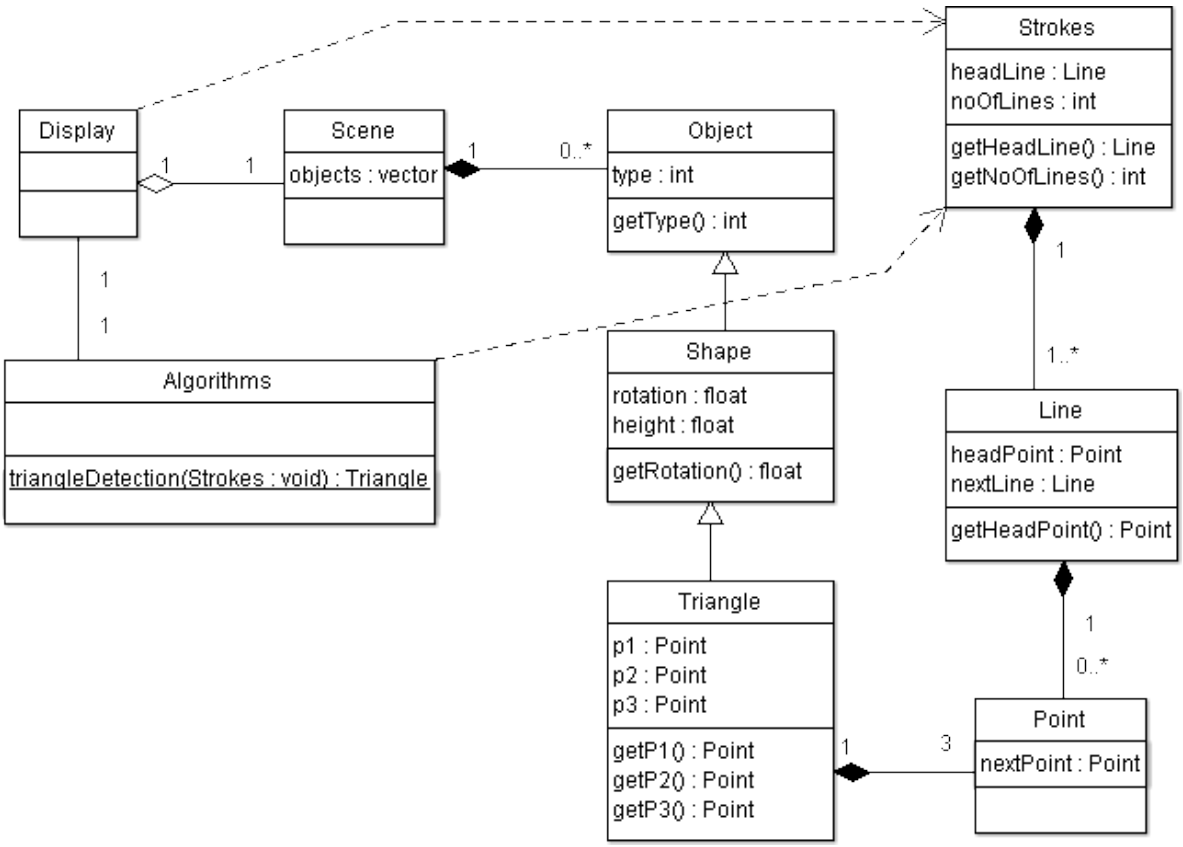


Figure 4.1: Simplified class diagram for Speedy Sketching

4.3 Shape Selection

In Speedy Sketching, users will frequently need to select shapes and objects for deletion and extrusion. This requires a projection of 2D coordinates on the screen to the 3D space for intersection tests, so the application can determine the user's choice. This is done by shooting a “ray” perpendicularly from the screen pixel and determining if the ray intersects with faces on either a 2D or 3D object.

The picking operation in OpenGL best suits this purpose. By drawing a small window around the mouse cursor, the scene surrounded by it is regenerated. A list of depth values of the objects in the window will be returned [25]. Speedy Sketching then selects the object that is closest to the screen.

4.4 Extrusion of Polygons

When extruding a shape, the face polygons on two ends are identical. To extrude properly, we need to know the normal of the original face, which should have been stored during shape creation. Then, when the user creates a height for the 2D shape, we will be able to project the points of the face to the pulled location. However, the operation is not complete, as for a solid shape, we need walls on its sides as well as the "roof" and "floor". For an n-sided polygon, the number of walls to be created will be n. See Figure 4.2 for an example.

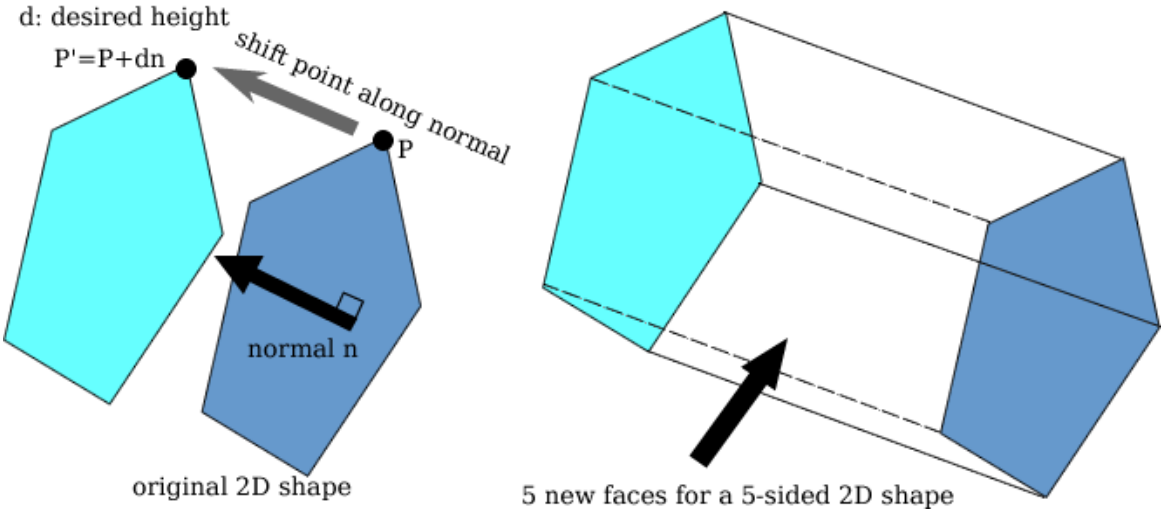


Figure 4.2: Pulling/pushing operation

In this way, any arbitrary 2D shape drawn by freehand draw can be extruded into a 3D prism. Care will be taken because OpenGL only guarantees correct rendering of convex polygons [20]. Tessellation may be required by calling the built-in method in OpenGL or using an external API. Samples of convex and concave polygons are provided in Figure 4.3. For a convex polygon, straight lines can be drawn from any vertex to any other vertex without leaving the polygon [21].

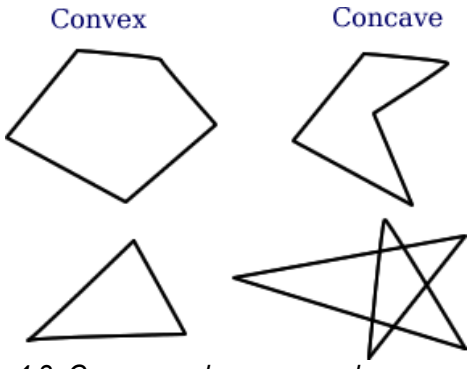


Figure 4.3: Convex and concave polygons

4.5 Drawing Common Shapes

The introduction of default shapes will speed up the creation process and improve accuracy, as users can rely less on freehand draw. To simplify the design, Speedy Sketching will be equipped with icons for default shapes. When the user clicks on them, the recognition engine will be expecting that particular shape. Then, points data sketched will be analysed to detect the orientation, length, width and other parameters. The following algorithms are all developed by the author. To reduce the time needed for development, all algorithms require that each stroke represents exactly one side or a multiple of sides of the shape. That is, one stroke cannot represent one and a half sides of a shape.

4.5.1 Mathematics

The shape detection algorithms will frequently require analysis and comparison of angles and distances between lines and points. The equations are introduced here but the mathematical proofs will be left to the readers. As Speedy Sketching exploits a drawing panel which is 2D, only equations regarding 2D geometry will be required.

(a) Distance between 2 points

The distance between 2 points is found by the Pythagoras' Theorem (Figure 4.4). It

is deduced by first calculating the differences in the x and y coordinates of the points, which are squared individually and their sum is found. The sum is squared rooted to find the distance. It will always be a positive number.

$$distance = \sqrt{(x1-x2)^2+(y1-y2)^2}$$

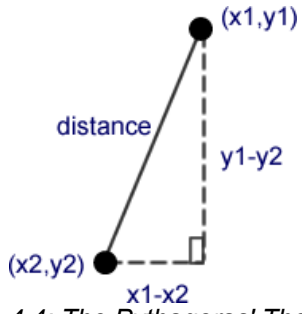


Figure 4.4: The Pythagoras' Theorem

(b) Distance from point to line [23]

The shortest distance of a point to a line, or the perpendicular distance, can be found by plugging the coordinates of the point into x and y of the equation of the line Ax+By+c=0. Taking the absolute value of the result, it is divided by the square root of the sum of squares of A and B. Figure 4.5 illustrates the perpendicular distance from a point to a line.

$$perpendicular\ distance = \frac{|Ax+By+c|}{\sqrt{A^2+B^2}}$$

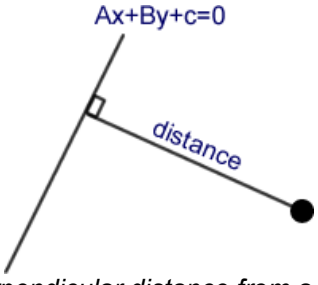


Figure 4.5: Perpendicular distance from a point to a line

Special attention is paid when the line is vertical, which has an undefined slope. Since the equation of the line is derived from the slope, a vertical line will result in zero division and failure to calculate the distance. Despite so, the distance

between a point and the vertical line is the distance between the x-coordinates of the point and the line, which turns out to be a simple subtraction. Again, the distance will always have a positive value.

(c) Angle between 2 lines [24]

The angle between 2 lines is related to their slopes, m1 and m2. The smaller slope is subtracted from the larger one, which forms the numerator. The denominator will be formed by adding one to the product of the 2 slopes. The arctangent of the resulting fraction is found, which will be the angle between the 2 lines. The angle will always be equal to or between zero and π/2 (Figure 4.6).

$$\theta = \tan^{-1}\frac{m2-m1}{1+m1m2} \quad where\ m2 > m1,\ 0 \le \theta \le \frac{\pi}{2}$$

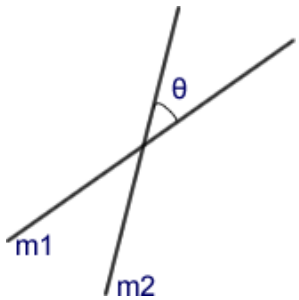


Figure 4.6: Angle between 2 lines

Again, this algorithm is affected by vertical line cases, which are dealt with carefully. If both lines are vertical, the angle is taken as zero. If one of them is vertical, we can find out the angle by subtracting the arctangent of the other slope from 90° (Figure 4.7).

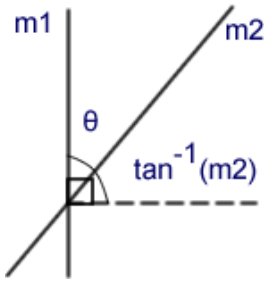


Figure 4.7: Angle between 2 lines when one line is vertical

4.5.2 Circle / Sphere Detection

The following method was originally suggested for detecting circles and spheres. Firstly, we can average all points to find the centre of the circle. With the centre found, we can estimate the radius by calculating the average distance from all points to the centre, using the Pythagoras' Theorem (Figure 4.8).

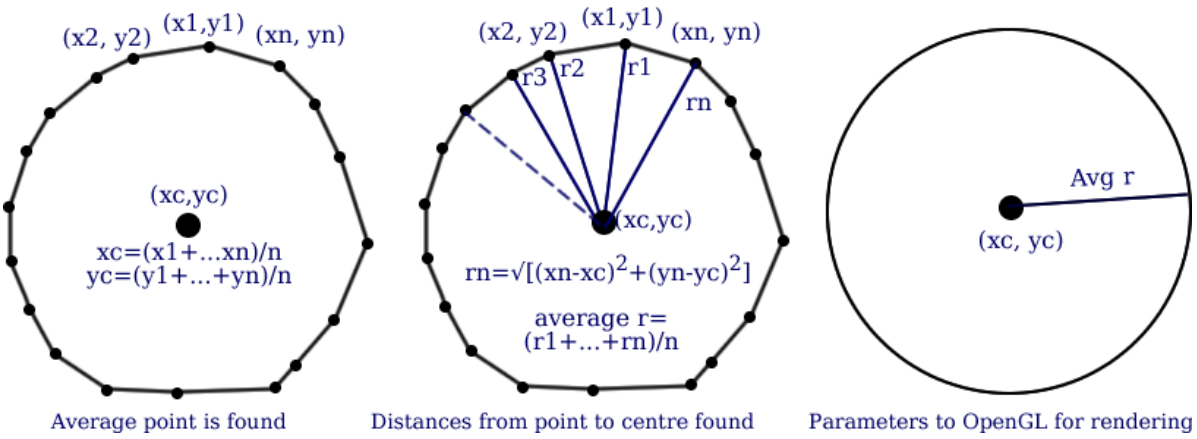


Figure 4.8: Original circle detection algorithm

However, during testing, it is found that the circle detected clearly suffers from a shift in position as shown in Figure 4.9. It is because the velocity of sketching varies in the following pattern: At the beginning of a sketched line, mouse velocity is zero, and it gradually speeds up. Then, mouse velocity reaches a maximum in the middle of a sketched line. When the user reaches the end of a line, they slow down to a halt. As most mice have a fixed polling rate, it results in a smaller number of points detected in a region when the mouse velocity is faster, and vice

versa. The increased number of points in the beginning and end points of a user stroke shifts the centre towards that region (Figure 4.10).

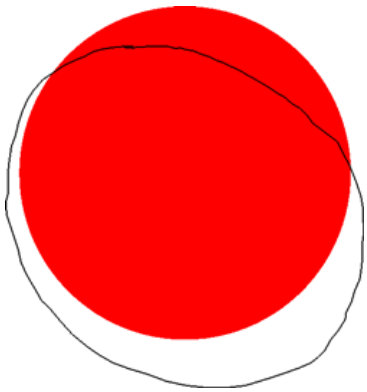


Figure 4.9: Sketched and detected circle before fix

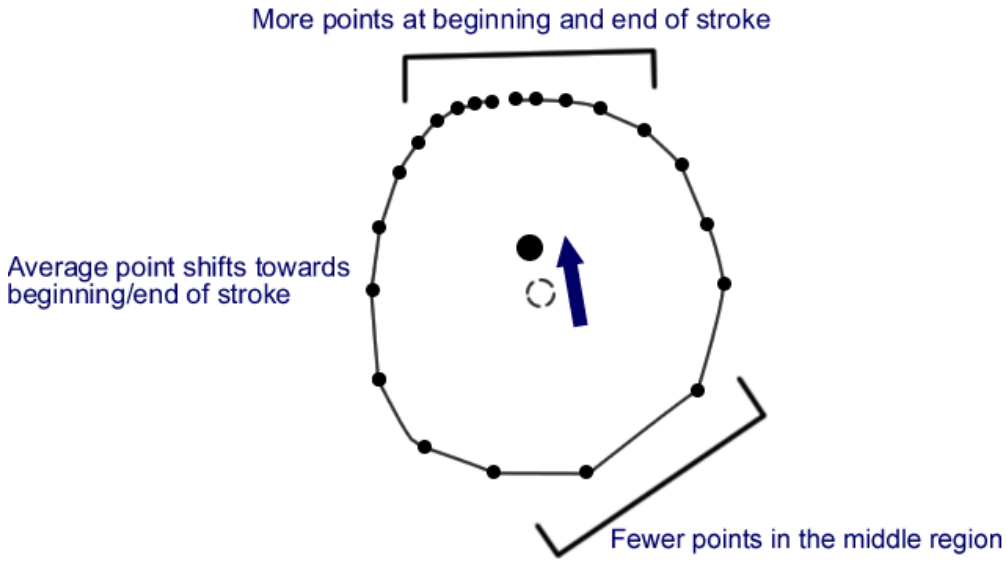


Figure 4.10: Shift in detected centre of a circle

To solve this problem, we have opted for the storage of a bounding rectangle as the shape is being sketched. As a bounding box remains unchanged regardless of point density, shifting does not occur. Moreover, a circle is supposedly symmetric in all directions, so we do not have to worry about the orientation of the rectangle. As shown in Figure 4.11, the algorithm now detects the centre of the circle properly.

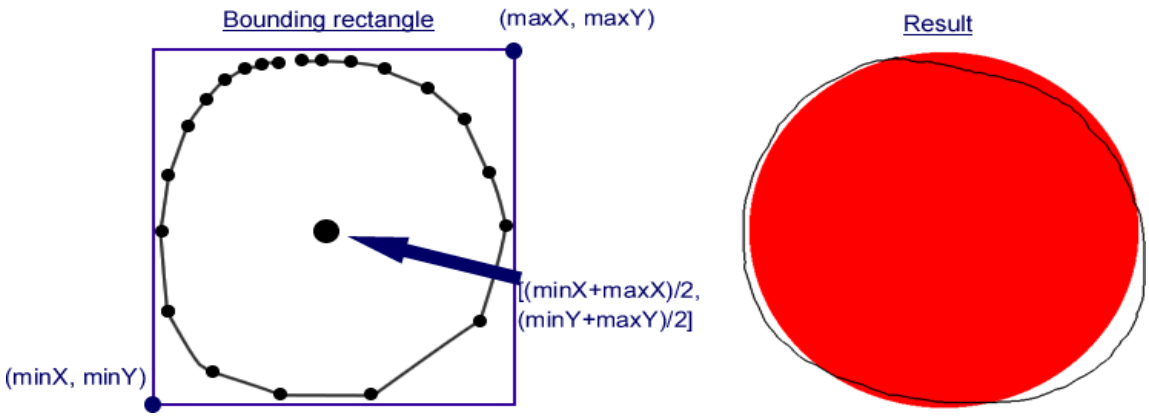


Figure 4.11: Bounding box for calculating the centre of circle

From this experience, it is determined that the method of averaging all points to find the centre is too crude, as it is affected by drawing speed. The author has then avoided using such method for other shape detection algorithms. Instead, interest points like corners are to be found from the sketched lines for more accurate detections. From the corners, we can work out the centre of the shape.

The insertion of a sphere is the same, except OpenGL is told to draw a sphere instead of a circle.

This approach of circle construction is different from that in Google SketchUp where users select the centre point and then drag out the radius of the circle. We believe this is a better method because it does not require users to estimate the centre of the circle which is difficult and allows them to draw naturally and freely like they are given a pen and a piece of paper, thus improving the learning curve.

4.5.3 Triangle Detection

For user convenience and variations in drawing habit, the user can draw a triangle in 3 different ways: 1 stroke, 2 strokes or 3 strokes (Figure 4.12). The algorithm correctly determines the triangle regardless of draw order and direction (clockwise or anti-clockwise, or a mixture of both). All three algorithms will be described below.

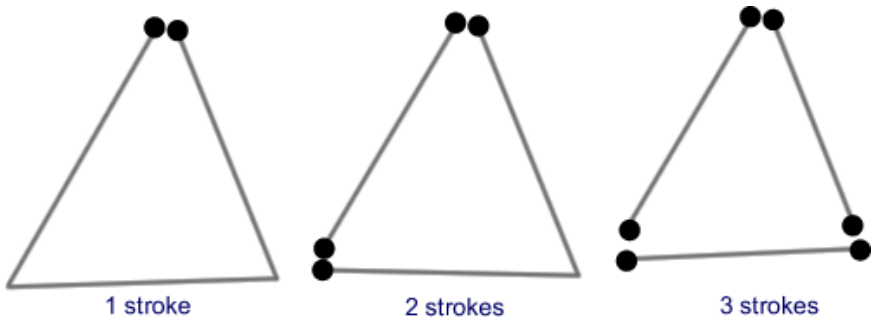


Figure 4.12: Possible scenarios for triangle detection

One-stroke detection

We have made the limitation that each stroke must represent exactly a side or its multiple. Therefore, the head and tail of the sketched line should be one of the intended vertices. They are averaged to find the mid-point, which is taken as Point A. In any triangle, the farthest point from any of the points must be one of the vertices. Taking the farthest point from Point A as Point B, we have 2 points of the triangle. To find the last point, we join A and B to form a line. The point with the maximum perpendicular distance from Line AB is Point C. The 3 vertices are then stored (Figure 4.13).

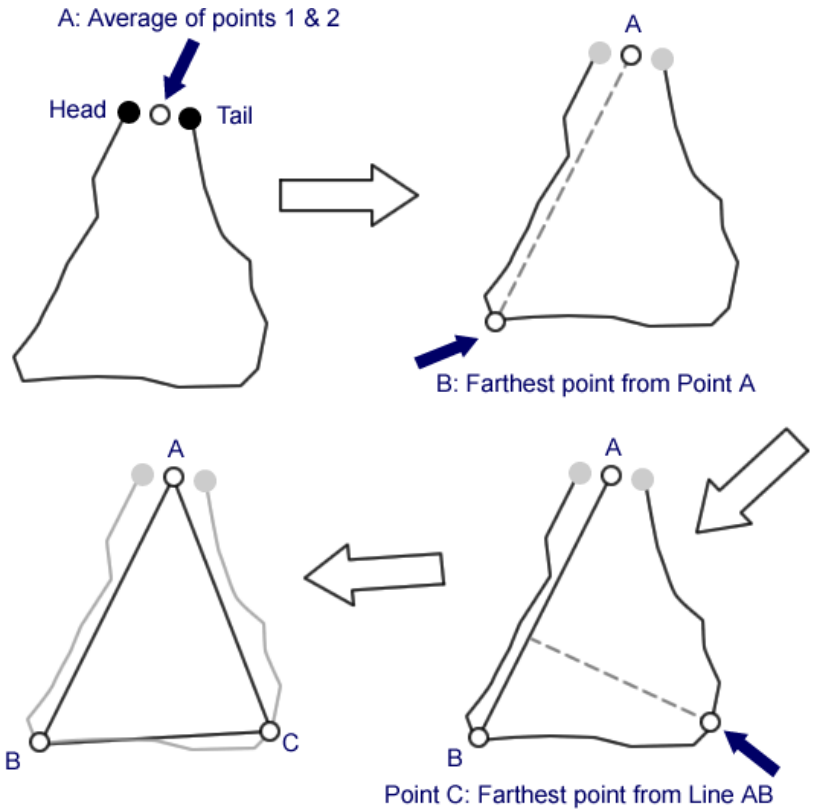


Figure 4.13: One-stroke triangle detection

Two-stroke detection

For two-stroke detection, one of the lines must be a straight line, and the other one shapes like a V. The first step of the algorithm identifies the two lines. First of all, the mid-point between the head and tail of each line, m, is found. Going through each line, the angle between the lines head point to m and m to each point is calculated, and the average angle found. The line with a higher average angle will be the 'V' line, and the other the 'I' line.

With the lines identified, it will attempt to find the 3 points of the triangle. From now on, it is similar to the one-stroke algorithm. The only difference here is having a total of 4 head and tail points, we can average 2 sets of closest points to find 2 points of the triangle directly. The 2 points are marked as A and B and point C is, like the former algorithm, found by calculating the farthest point from line AB. This is illustrated in Figure 4.14.

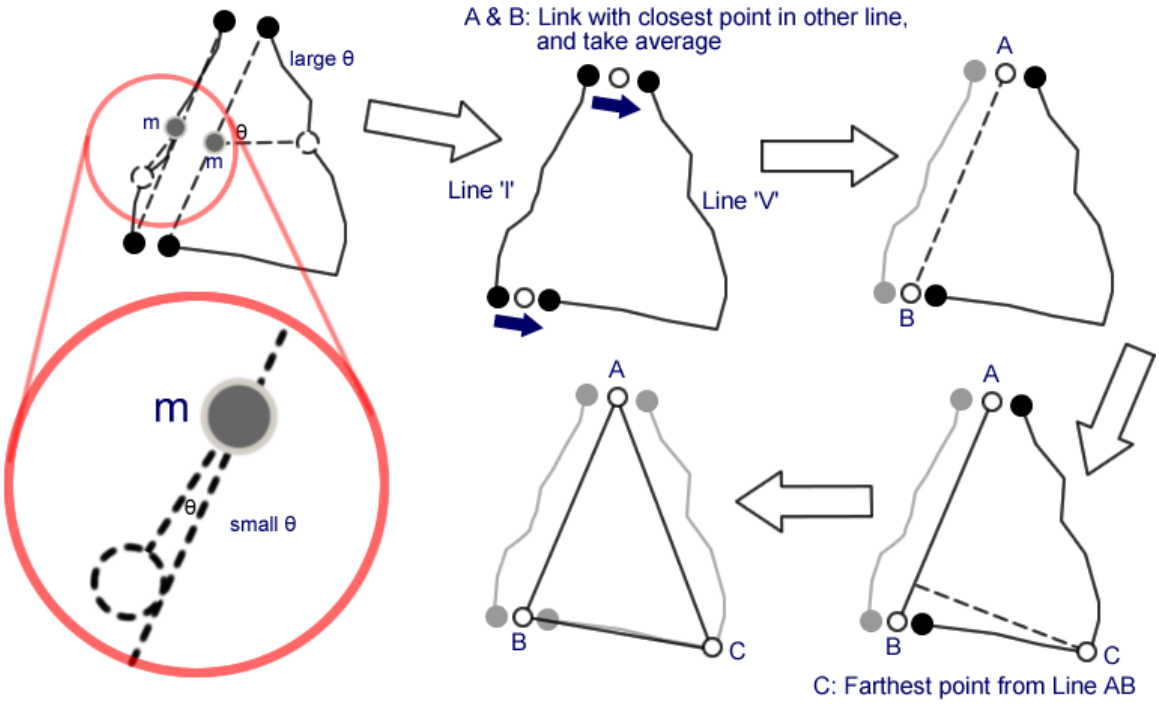


Figure 4.14: Two-stroke triangle detection

Three-stroke detection

This algorithm is the simplest of all. Going through all the 6 end points in the 3 strokes, the closest one to itself is assumed to be the same point. Sets of 2 points are averaged and each set will result in a point in the triangle (Figure 4.15).

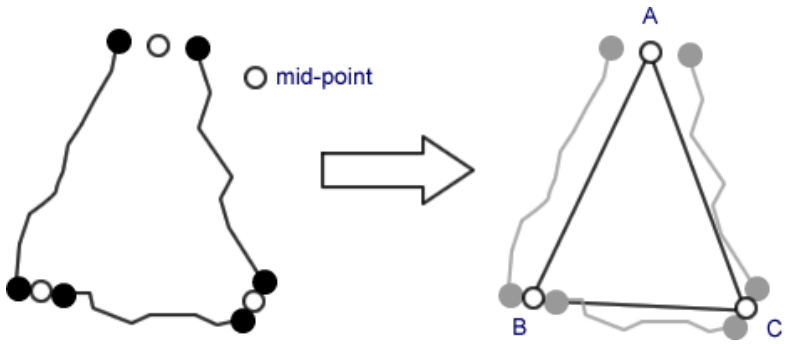


Figure 4.15: Three-stroke triangle detection

4.5.4 Cone Detection

Cone detection is based on the triangle detection algorithm as the user will be drawing the desired cone as a triangle. Therefore, the cone can be drawn in 1, 2 or 3 strokes. The algorithm first calls the triangle detection algorithm which

returns a triangle represented by 3 vertices in space. The 3 angles are compared to find the closest 2 angles, which are treated as the base of the isosceles triangle. As a user sketch is never perfectly isosceles, the 2 angles are averaged to find the angle of the "isosceles triangle" in the cone. The inclination of the base of the triangle is found by finding the angle between the base line and a horizontal line with a zero slope (Figure 4.16).

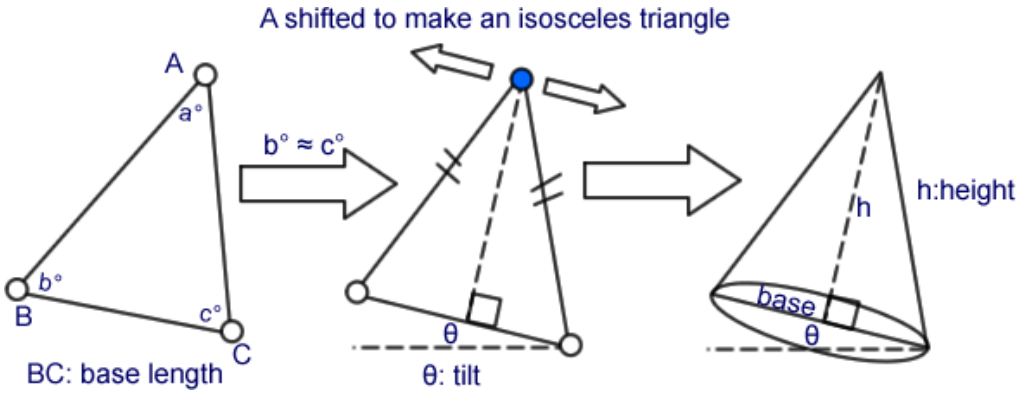


Figure 4.16: Cone detection algorithm

However, the inclination alone is not enough. For each base angle, there will be 2 orientations of the cone, which are above the line and below it. To solve this, the top point of the isosceles triangle will be plugged into the equation of the base line. If the result is bigger than zero, the point is above the line (Figure 4.17). Otherwise, it is below, which means the cone is pointing downwards. In this case, the cone will need to be rotated by 180° in addition to the original inclination. As a result, the cone object has an extra boolean to store the orientation.

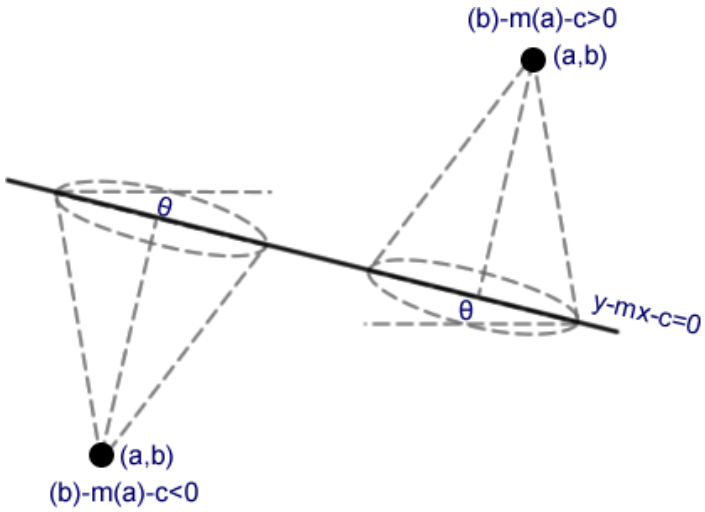


Figure 4.17: Differentiating between 2 types of cones

4.5.5 Rectangle / Cylinder Detection

It was originally determined that the centre of a rectangle can be found by averaging all user strokes (Figure 4.18). As discussed in Section 4.5.2, such algorithm is affected by drawing speed. The author also planned to use the angular tolerance algorithm to reduce the points to the 4 corners of the rectangle. Again, this is almost impossible given the following dilemma:

As a user stroke is noisy, lines can be zigzagged. To discard the highest number of points possible, a *high* angular tolerance has to be set. In contrast, the application cannot ask the user to provide perfect ninety-degree corners or it will defeat the purpose of *Speedy* Sketching. The user may draw the rectangles quickly resulting in round corners. To prevent the algorithm from discarding a rounded corner of a rectangle, a *low* tolerance will be needed.

As it is extremely difficult to set a tolerance value, and that a tolerance value varies among users, we are very likely to end up with a distorted rectangle (Figure 4.19), from which we cannot determine the parameters.

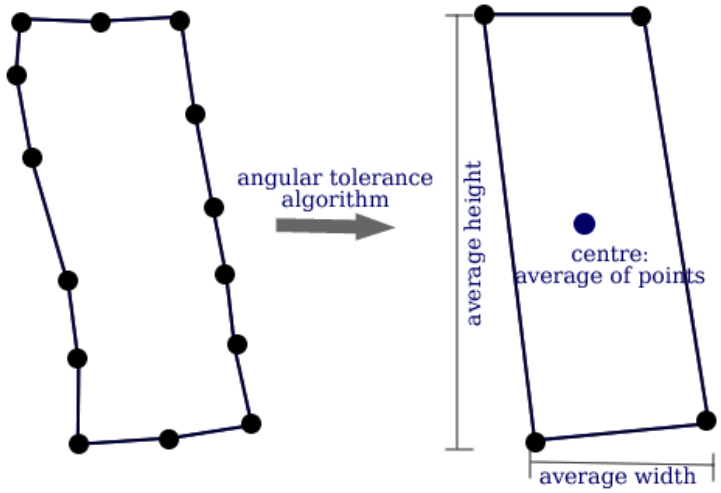


Figure 4.18: The original rectangle detection algorithm

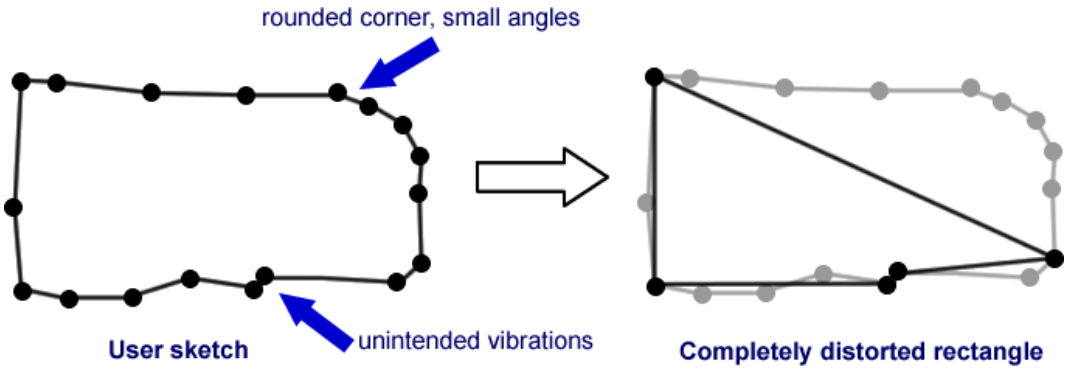


Figure 4.19: Rectangle detection with angular tolerance algorithm

Due to time constraints, rectangle detection is not fully implemented, and the user must draw it with 2 strokes. However, like the triangle detection algorithms, the user can draw the strokes in any orientation or order (Figure 4.20).

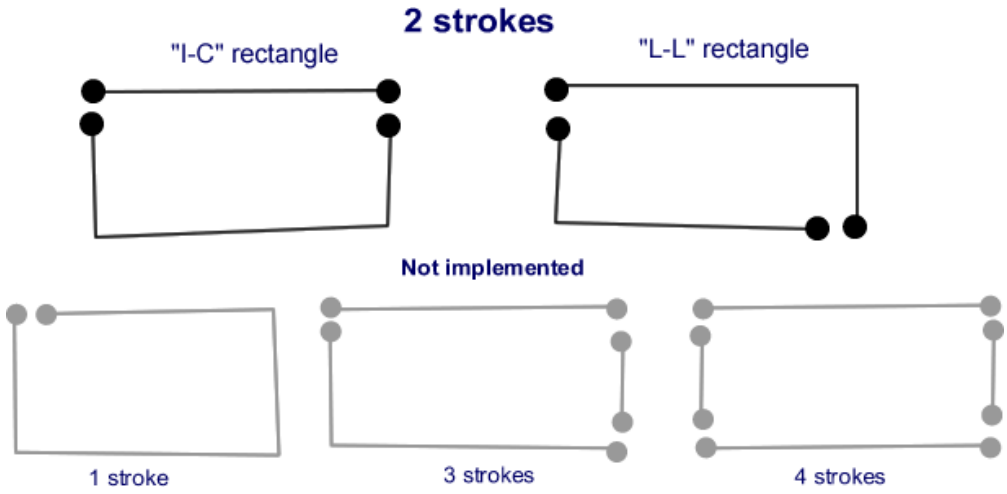


Figure 4.20: 2-stroke rectangles implemented

Like triangle detection, the first step of rectangle detection lies in identifying the 2 types of 2-stroke rectangles, namely, the "I-C" rectangle and the "L-L" rectangle. For each line, the head and tail are joined to form a line. Then, for each point in each line, it is linked with the head point of the line to form another line. The angle between them is stored and the averaged angle for the line calculated. For the "L-L" rectangle, the difference between the 2 average angles will be small, as the "L"s will have similar shapes. However, for the "I-C" rectangle, the average angle for the straight line will be much smaller comparing with the angle for the "C" (Figure 4.21). The algorithm checks whether one of the average angles is bigger than 5 times the other one. Using this difference, the rectangles are identified and the respective algorithms called.

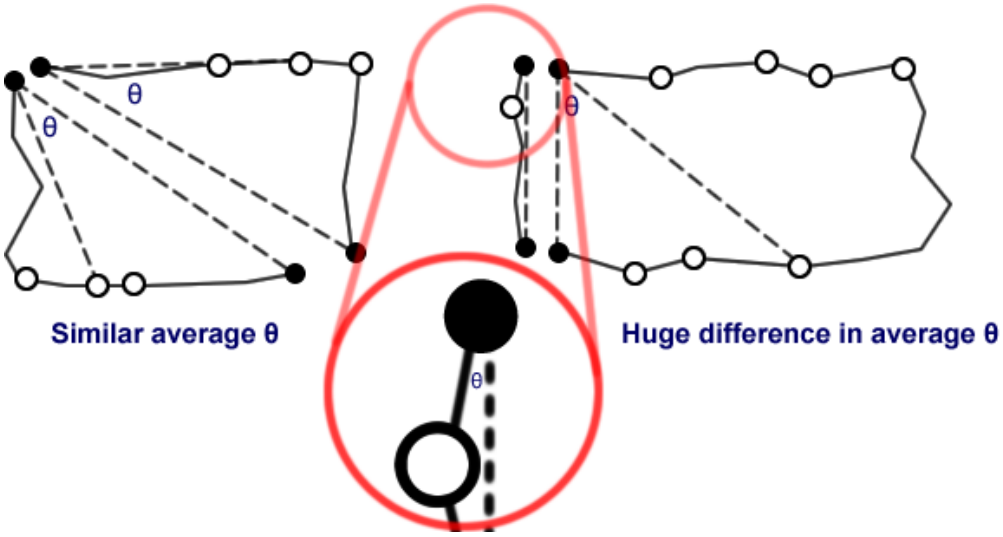


Figure 4.21: Differentiating between "I-C" and "L-L" triangles

For both types of rectangle, the aim is to find out 4 points in space for further determination. The methods are extremely similar to the triangle detection ones. For an “I-C” rectangle, the available 4 head and tail points are put together in 2 groups of 2 and averaged to find 2 points, A and B. Then, the farthest points from A and B – which will be on the diagonal lines – are found on the “C” line, giving a total of 4 points. For the “L-L” lines, averages are again taken to find 2 of the points, which are opposite corners of the rectangle. Then, each half of the rectangle will be a triangle itself, and the maximum perpendicular distance from the head-tail line will be another corner on the rectangle (Figure 4.22).

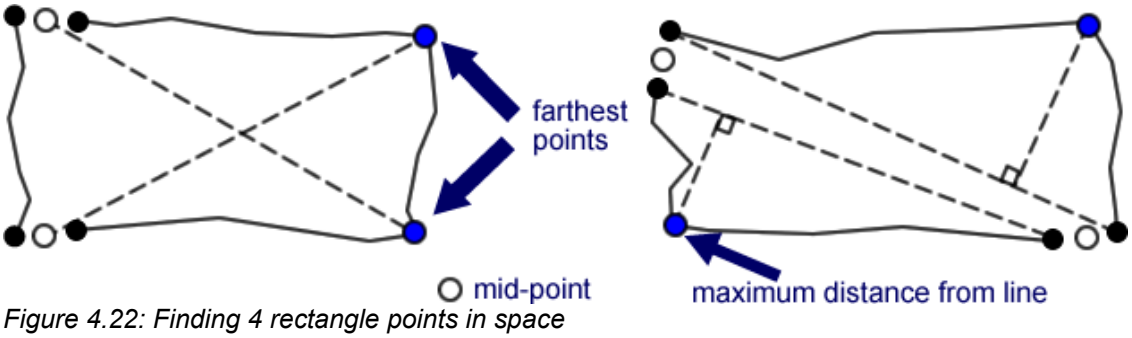


Figure 4.22: Finding 4 rectangle points in space

After the above, we have obtained 4 points in space, each denoting a corner on the rectangle. As a rectangle has a fixed width, length and 90-degree corners, some generalisation is required with information contributed by all 4 points. The

centre can be determined from the average of them. As the 4 points are actual corners previously detected, the centre will not be shifted by variations in point density. With the 4 points, the average length and width can also be calculated.

Originally, the orientation of the rectangle is assumed to be the average contribution from all 4 lines. (Although 2 of the lines are perpendicular to the other ones, their contribution can be averaged because we can calculate the contribution in the perpendicular direction by the formula: *slope x perpendicular slope = -1*).

However, this has been adjusted because the shorter sides of the rectangle tend to be more inaccurate. A human user will tend to think that the longer edges are the "dominant edges" of the rectangle and pay more attention to them, because they are more distinct to the human eye. Also, the human user will often find it harder to draw shorter lines given the same amount of random vibrations in a human hand. The algorithm now only calculates the orientation of the rectangle from the longer edges, and this has been found to give more desirable results. The formula is similar to the one used in cone orientation. The long edge is taken as the base and its angle between a horizontal line determined.

The cylinder detection is based on the rectangle detection algorithm. With the length and width of the rectangle known, this can be converted to the height and radius of the base circle. For a "drum cylinder", the algorithm swaps the length and width.

4.5.6 Freehand Draw

As mentioned in Section 2.3.2, line generalization techniques are needed for reducing data to be processed by an application, and to provide an aesthetically appealing appearance of shapes. As freehand draw includes any form of data including curves and straight lines, the points data can quickly become very large.

For these reasons, freehand draw lines are smoothed and then simplified by the following algorithms as soon as the user releases the draw button.

McMaster's Slide Averaging Algorithm [9]

This algorithm has low time and space complexities as it is only taking the average of every 5 points and sliding the third point halfway towards it. The new value of the third point is saved until all points are considered. Figure 4.23 explains the Slide Averaging Algorithm in detail. Using this algorithm, the number of memory locations required will be twice the length of the line – O(N). Likewise, the number of divisions will be equal to the number of points minus the 4 end points – O(N).

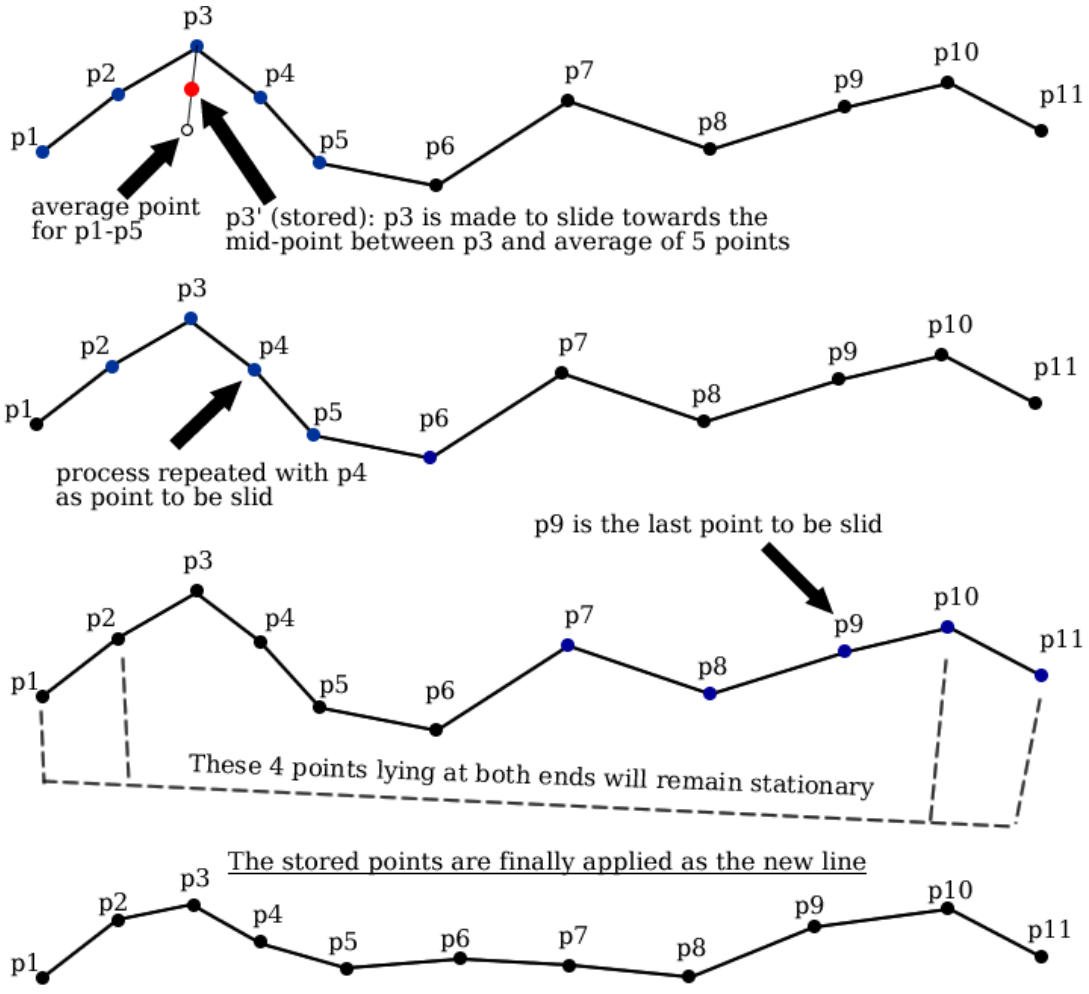


Figure 4.23: McMaster's slide averaging algorithm

Angular Tolerance Algorithm [9]

Although the above algorithm can smooth lines, it is unable to reduce data size. Angular tolerance algorithm solves this problem by connecting point n with point n+2, and finding the angle between lines n/n+2 and n/n+1. If the angle is smaller than a tolerance value, then point n+1 does not make an impact on the shape of the line and it will be removed. Otherwise, the point will be retained. By setting a high tolerance value, we can eliminate unwanted fluctuations from the line and hence the amount of data is effectively reduced.

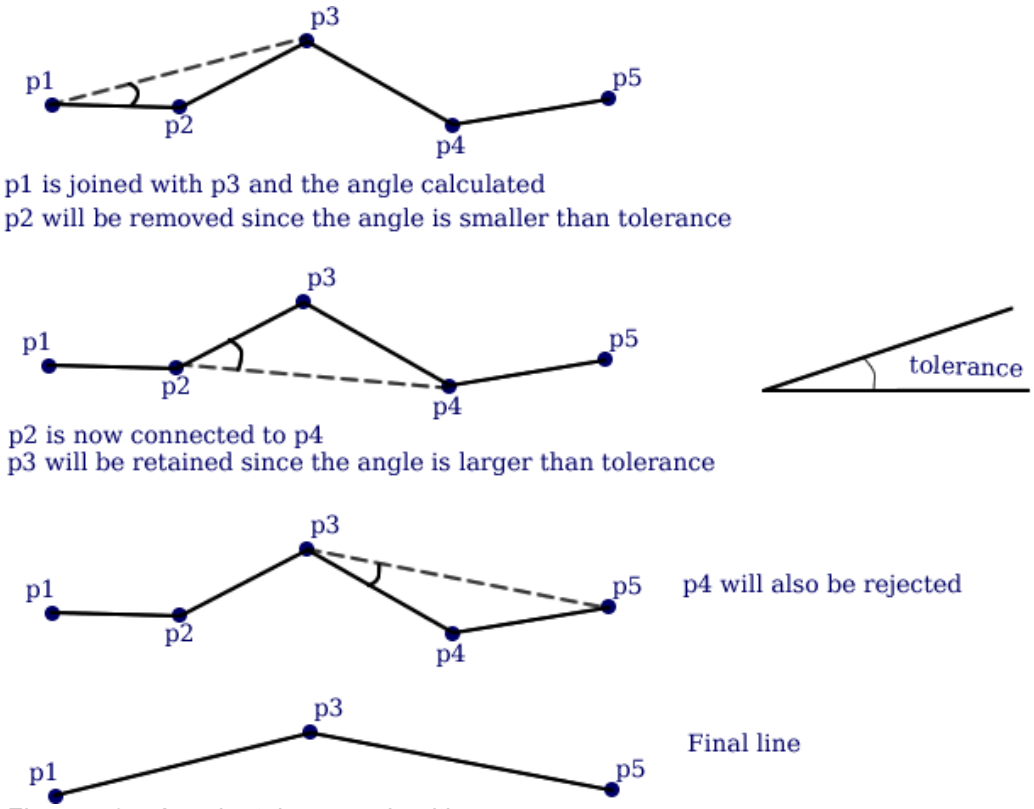


Figure 4.24: Angular tolerance algorithm

We have just covered the algorithms developed or used by the author for quickly detecting and generating shapes from user strokes. The results will be presented in the next chapter, and the accuracy of the algorithms determined.

5 Results

The application is compiled as a Windows executable file (.exe) so it is only available for Windows users. However, given the multi-platform nature of Qt and OpenGL, we can safely assume that the application can be converted to support other platforms easily.

5.1 General Robustness

The application is made to tolerate irregular behaviour. For example, a user may change the shape detection method or attempt to move the scene in the middle of drawing. Also, a user can draw random lines instead of a detectable shape, causing the algorithms to output abnormal looking shapes. These kind of unpredictable behaviour are well caught by the system and will not cause the program to crash. However, we have only tested the application as a developer. In the next chapter, the application will be tested by a number of invited users, who will provide even more unpredictable behaviour to test the robustness of the system.

5.2 Graphical User Interface

Adhering to the design principles, the interface is very simple and all the possible operations are executable in one click. Shape selection icons are large and easy to click. The shortcuts are added to the image as well as the tooltip, because during one of the testing phases, it is found that a user may have difficulty holding a pen steady for the tooltip to appear. Also, casual users of the computer do not tend to remember shortcuts, rendering them useless. The numbers on the shape selection icons also allow them to select shapes on the

keyboard with one hand and draw with another holding a pen or a mouse, without having to memorize the shortcuts. The menu bar is shown in Figure 5.1. From 1 to 8, they are the freehand draw, circle, triangle, rectangle, sphere, cone, tubular cylinder and drum cylinder detections. 0 refers to the extrusion operation.

Figure 5.1: The menu bar

The user can press Ctrl+N and Del icons to clear the scene and change to delete mode respectively. It should be noted that *Undo* (Ctrl+Z) only deletes the last object drawn unlike a traditional Undo button, which undoes any action. The *Panel in*, *Panel out*, *Zoom in* and *Zoom out* icons are located on the bottom right corner.

Comparing with professional packages, Speedy Sketching features a very simple and uncluttered interface. It does not have a cascaded list of menus which casual users are unlikely to go through. The Drawing Panel shows clearly 2 rotational axes at all times, so users are likely to take them into account when drawing. The final version of the GUI design is shown in Figure 5.2.

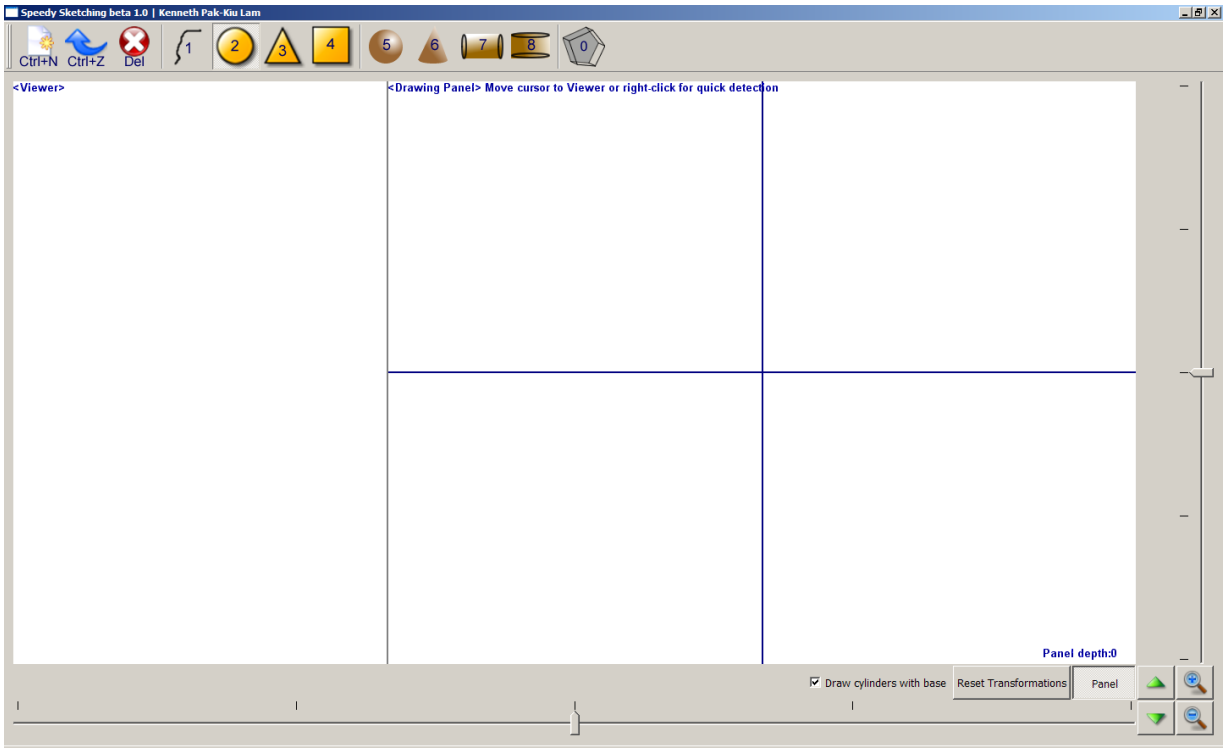


Figure 5.2: The final GUI design

5.3 Shape Detection Algorithms

It should be noted that for the purpose of this report, previously sketched lines are adjusted to remain black, while they turn grey in the application.

5.3.1 Circle / Sphere Detection

The circle detection algorithm manages to find the centre and desired radius of the sketched circles correctly given a noisy line. This does not seem to change regardless of the size of the circles or spheres, which is encouraging (Figure 5.3).

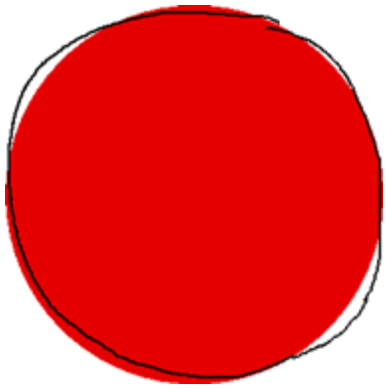


Figure 5.3: Circle detection algorithm

The algorithm is able to make sensible responses to random inputs provided by the user, shown in Figure 5.4. This is particularly useful when a user wants to draw a lot of randomly sized circles but does not want to sketch each one of them carefully, which is time consuming. This again enhances the speedy of sketching.

Finally, the user may want to provide more points data to get a more accurate radius and centre. This can be achieved by drawing lots of circles repeatedly until they eventually release the draw button (Figure 5.5).

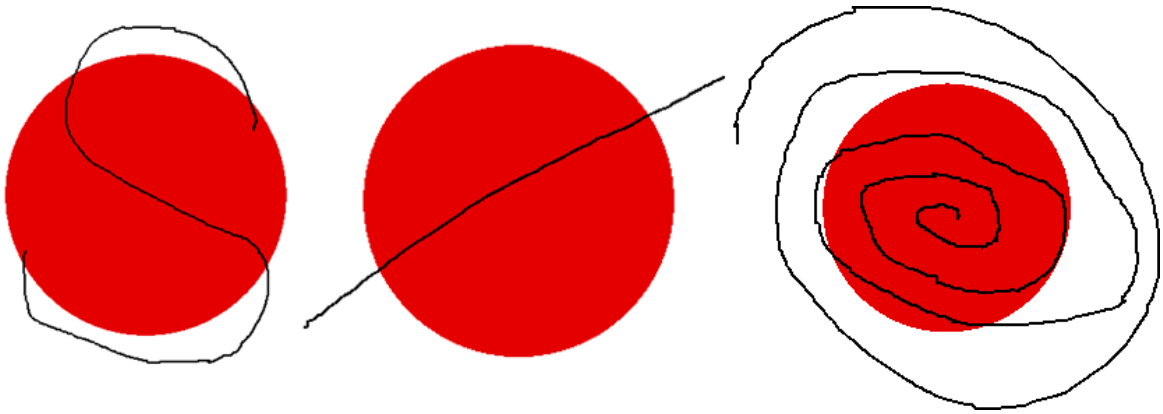


Figure 5.4: Random inputs and circles generated

Figure 5.5: Providing more points data for circle detection

The above examples hold true during the span of the development and testing. Therefore, we believe that the circle / sphere detection algorithm is robust enough to appear in a commercial application.

5.3.2 Triangle Detection

Similar to the circle detection algorithm, the triangle one is unaffected by the size of the triangle. The differences between 1-stroke, 2-stroke and 3-stroke algorithms are also extremely small (Figure 5.6). This is important in the sense of human-computer interaction, because it means the user experience remains unchanged even for different user behaviour and drawing scenarios.

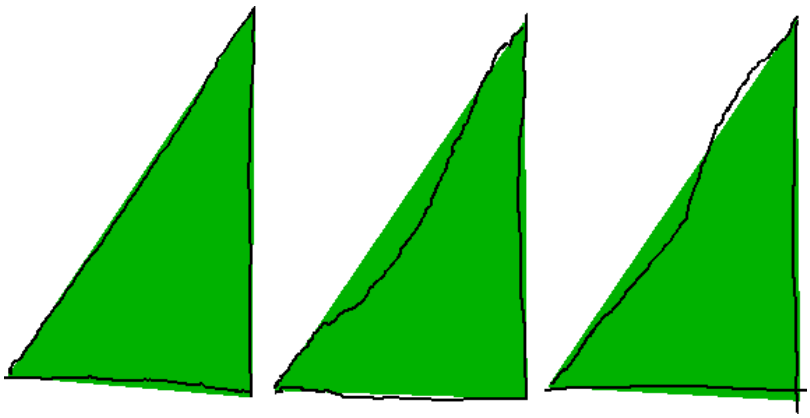


Figure 5.6: One, two and three stroke triangles

For extreme cases, e.g., a long and thin triangle, a user may find it hard to draw it by hand on a piece of paper. It is even truer for Speedy Sketching where users

can be drawing with a mouse, which is relatively clumsier. But this does not seem to affect Speedy Sketching, where a triangle is successfully detected. As shown below in Figure 5.7, even if 2 sides of a sketched triangle overlap before reaching a vertex, little fluctuation from the sketched strokes is spotted.

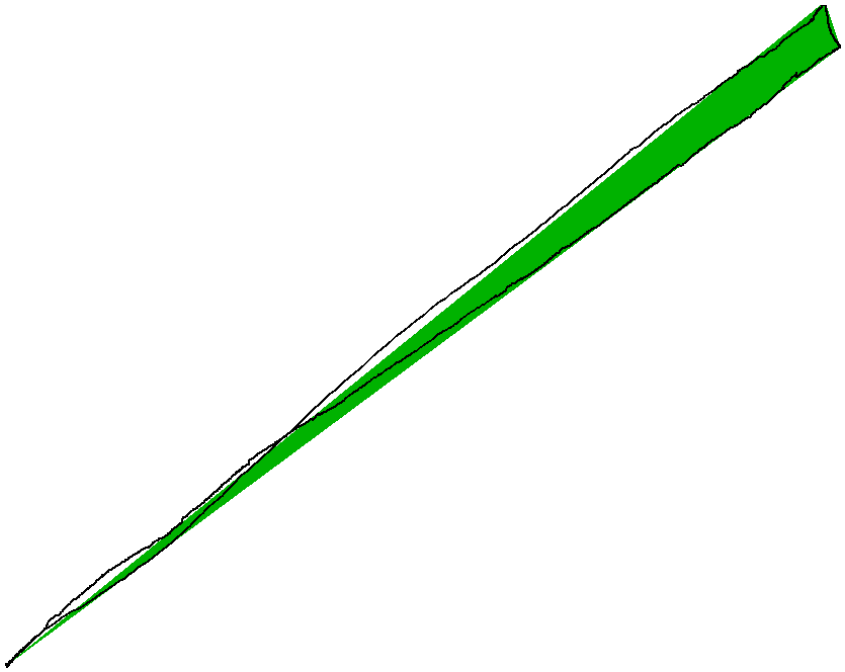


Figure 5.7: Detection of a long and thin triangle

As with circle detection, random inputs from the user will not crash the system in any way (Figure 5.8). However, in the middle image shown below, we can see that a straight line is interpreted as an extremely thin triangle. This is probably a case where the user activates the detecting algorithm accidentally before finishing the shape. To improve the algorithm, we may determine the area of the triangle in terms of screen coordinates. If the area is lower than a threshold, we can discard the triangle and tell the user to redraw it.

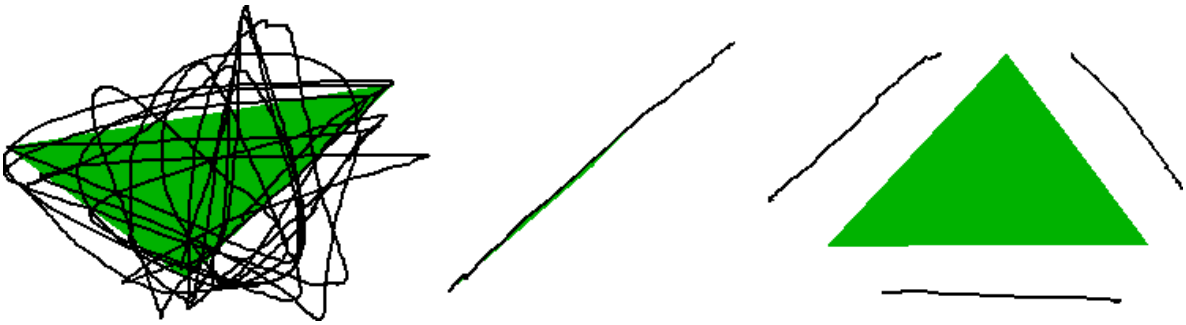


Figure 5.8: Random inputs and triangles generated

Overall, the author feels that the triangle detection algorithms are extremely robust. Although it seems to complicate the problem of specifying 3 vertices of a triangle (which can be done by just pointing and clicking the 3 vertices), the user can use a consistent drawing method throughout the entire application, which reduces the time required to learn the interface.

5.3.3 Cone Detection

Although the cone detection algorithm is based on triangle detection, a cone requires the triangle to be isosceles, which is rarely the case for a noisy user sketch. The algorithm fixes the base and slides the tip of the triangle to make it isosceles. For this reason, the orientation of the cone may seem a little off (Figure 5.9). To improve the algorithm, we may fix the top of the cone and adjust the triangle by moving the base points.

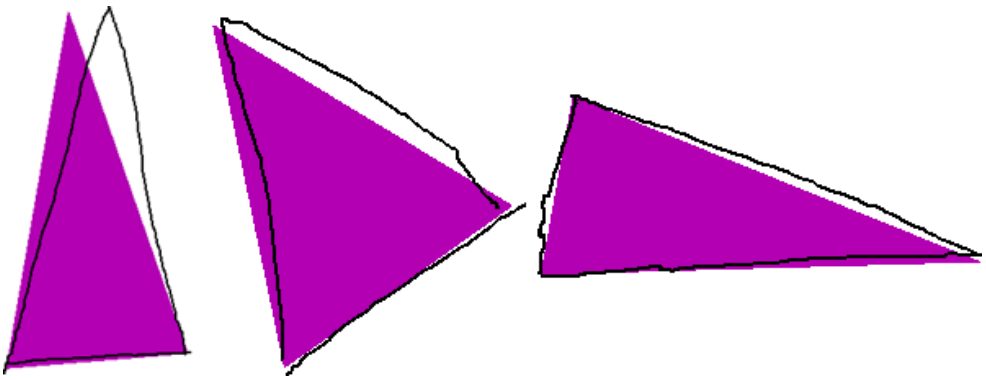


Figure 5.9: Cone detection algorithm

A second problem will be the detection of almost equilateral triangles, because

it is an isosceles triangle in all 3 orientations. With noise present in their strokes, the users may not be able to draw the desired isosceles triangle. In Figure 5.10 below, the user is attempting to draw an upright cone, but rotations will show that the tip is pointing downwards. This problem is currently unsolved in the application. Also, due to time constraints, a perfectly equilateral triangle is ignored by the application and no shape is returned. (Though due to the grid nature of the screen, such triangle may hardly be drawn.)

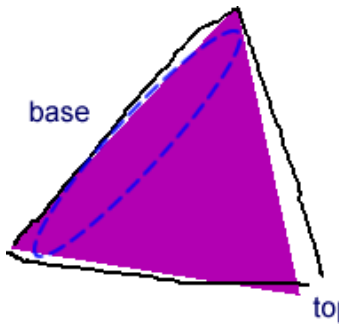


Figure 5.10: Detection of an equilateral triangle

The equilateral triangle problem reveals that more intelligence is needed in sketching systems where ambiguity exists. Despite the fact that a sketch can represent a lot of information, supplementary data is often needed for processing. For example, if all 3 angles of a triangle are too similar, the user may be prompted for the desired orientation of the cone. Furthermore, if voice recognition is added to a sketched system, the user can specify the tip of the cone while drawing the shape, thus saving time.

All in all, we believe that more investigation is needed to solve the orientation and equilateral triangle problems.

5.3.4 Rectangle / Cylinder Detection

As rectangle and cylinder detection algorithm are the same, their results are combined to one section. The detection is generally acceptable when lines are straight and angles are sharp (Figure 5.11). The use of the longer edges to

determine the orientation means that the result is unaffected by very small and noisy short edges. In the right rectangle in Figure 5.11, one of the shorter sides clearly deviates from the main desired rectangle, but the rectangle remains in the correct orientation.

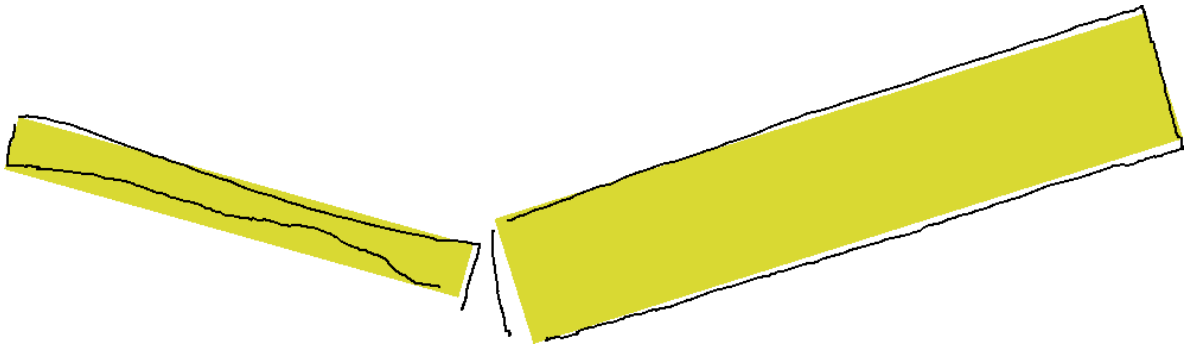


Figure 5.11: "L-L" and "I-C" rectangles detected in different orientations and sizes

One of the problems faced with designing the algorithm is the tendency for the users to draw a parallelogram instead of a rectangle, as demonstrated in Figure 5.12. Using the longer edge for orientation, the user will feel a more accurate detection. However, the current algorithm does not worry about the orientation of the skewed short edges, and the slanted length of the short edges will be treated as the width of the rectangle. As seen in the same figure, the width of the rectangle appears to have widened. The deviation of the width is related to amount of tilt of the shorter edges from the perpendicular line coming out from the longer edges.

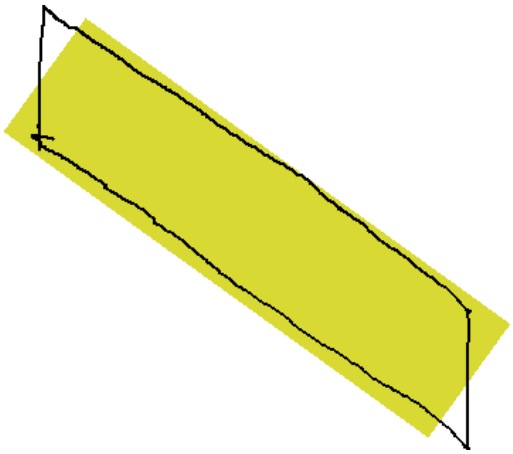


Figure 5.12: Converting a parallelogram to a rectangle

For long and thin rectangles, the result is also fairly acceptable (Figure 5.13). It should be noted that the user lines are not straight, leading to the slight discrepancy observed.

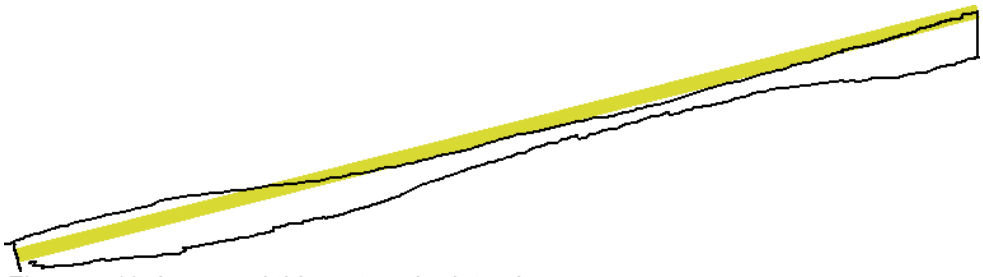


Figure 5.13: Long and thin rectangle detection

Apart from the parallelogram effect, the rectangle detection algorithm is relatively less accurate than the other algorithms, and more work is needed. It is notable that sometimes the wrong algorithm (“I-C” / “L-L”) is used for detecting the rectangle if the user sketches a trapezium. This will lead to deviations in orientation, length and width (Figure 5.14) and usually a long and thin rectangle is returned.

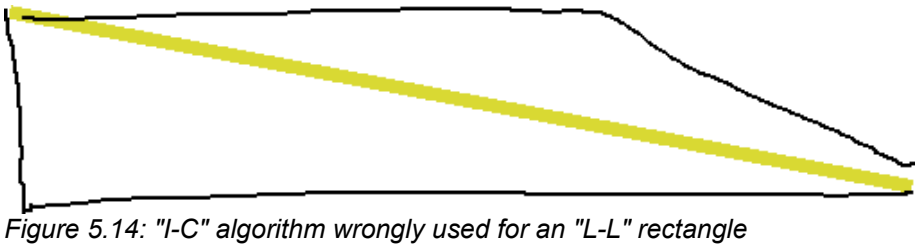


Figure 5.14: "I-C" algorithm wrongly used for an "L-L" rectangle

From the above problems, we can see that the rectangle detection and hence the cylinder detection is still fairly premature. The algorithm to identify “I-C” and “L-L” rectangles is to be revamped, and resistance towards wrong inputs upped. For now, users are required to draw rectangles with extra care and precision to prevent the above problems, which violates the philosophy of *Speedy* Sketching.

5.3.5 Freehand Smoothing and Simplification

The freehand smoothing is default to one iteration of McMaster's slide averaging algorithm and an angular tolerance of 1.5°. The algorithm has been tested by drawing the capital letters A-Z, which contain a good number of straight and curved lines, on the screen. Using a pen interface with quite a low polling rate, this algorithm is found to reduce the number of points by 42.9%. A laser mouse with a high polling rate creates about 5 times the number of points but this algorithm manages to reduce the number of points by 66.2%.

5.3.6 Limitations in All Algorithms

Due to the amount of time available for design and development, the shape detection algorithms have been simplified. While they currently work well, there is room for improvement. Firstly, the algorithms never take intersecting lines into account. Some users like to draw longer sides and define a shape by the intersecting points of the lines (Figures 5.15). However, this is not supported by Speedy Sketching, as we take the approach of averaging the available lines and end points.

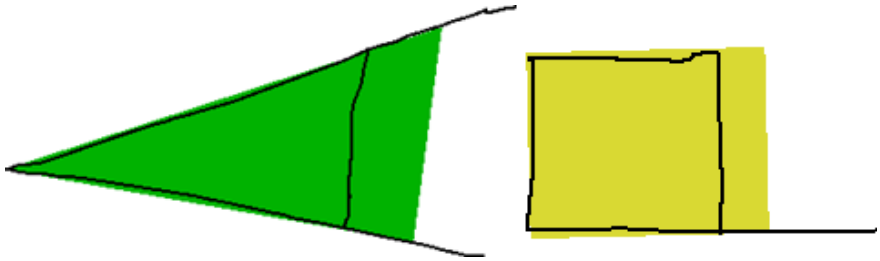


Figure 5.15: Intersecting points of a triangle and rectangle ignored

Moreover, there is currently no intelligence to tell if a shape is complete. If a user draws a triangle with 1 or 2 strokes, they will have to activate the detection algorithm themselves. For a new user, this is potentially confusing because the application seems to be ignoring the sketched shapes. If we have a way to determine whether a shape is complete, then the algorithms can be activated automatically, saving time and removing another obstacle towards a user-friendly application.

5.4 Extrusion

Due to time constraints, only extrusions for triangle and rectangle have been implemented. The operation is quite limited because the user can only drag out the height of the shape relative to the original 2D sketch. However, we believe this is a great addition to Speedy Sketching as 2 more kinds of shape can be drawn, i.e. rectangular and triangular prisms. We will demonstrate extrusion by drawing a house. Firstly, the user draws a cylinder, a triangle and a rectangle (Figure 5.16).

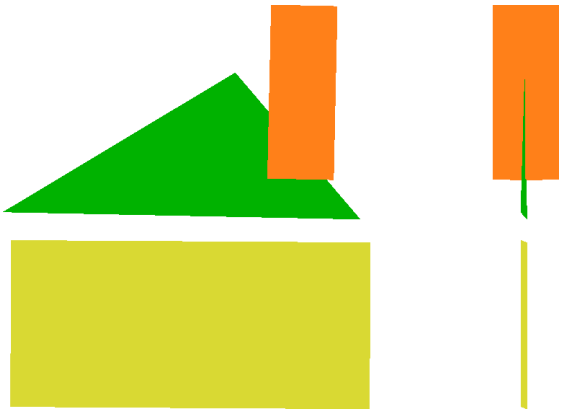


Figure 5.16: Front and side view of the house before extrusion

Then, the user selects the extrusion icon and drags out the height (depth) of the rectangle and triangle, by holding the left mouse button and moving to the right.

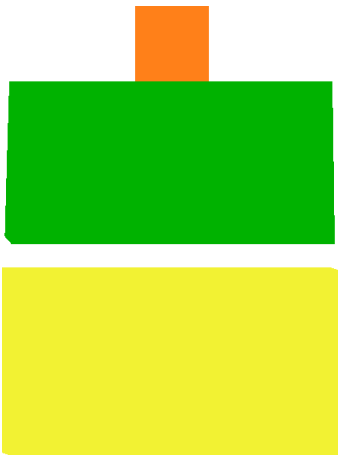


Figure 5.17: Side view of house after extrusion

The final side view is shown in Figure 5.17. Two more views in perspective mode have been displayed to show the depth of the objects (Figure 5.18).

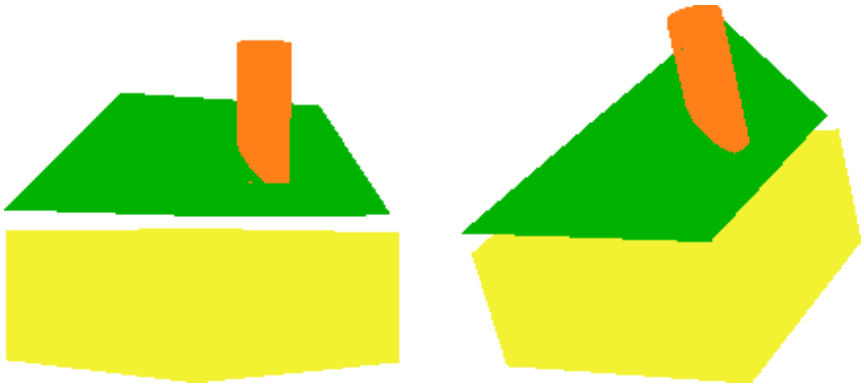


Figure 5.18: Screenshot of the house from 2 different angles

5.5 Drawing with Speedy Sketching

To demonstrate the usability and intuitiveness of Speedy Sketching, we will present the entire process of drawing a simple scene containing a luxo lamp and several spheres. The example below will also show the workings of rotation and movement of the Drawing Panel.

First of all, the user clicks the cone icon and draws an isosceles triangle near the Y-axis of the Drawing Panel. Doing this ensures the object will rotate approximately around its own centre (Figure 5.19).

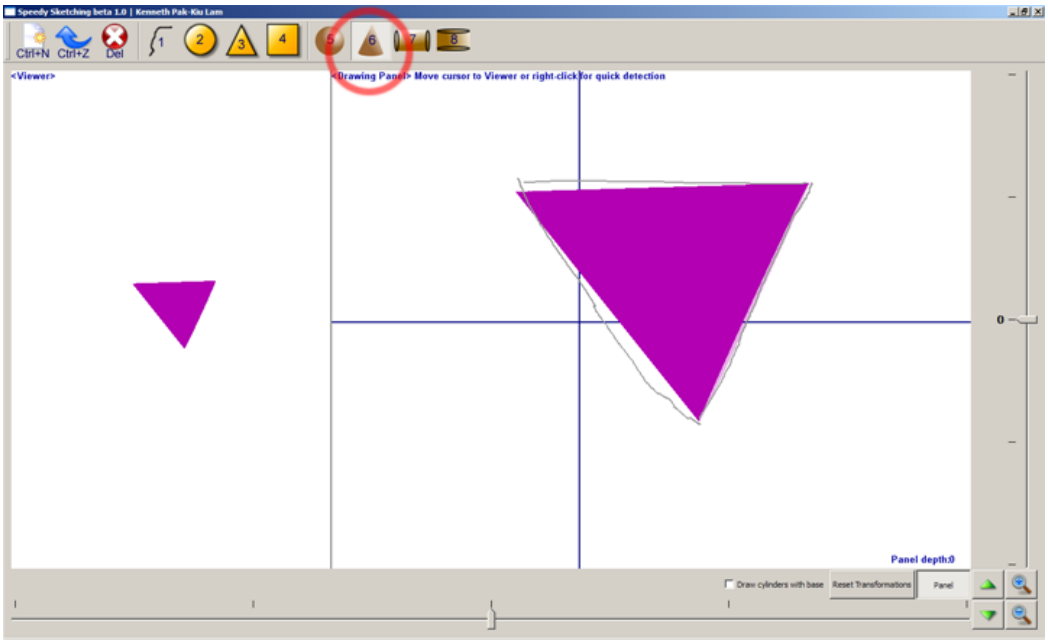


Figure 5.19: Drawing the shade of a luxo lamp

Then, they select sphere mode and draw a circle on the screen. The circle overlaps the cone, which results in the sphere being embedded in the cone (Figure 5.20).

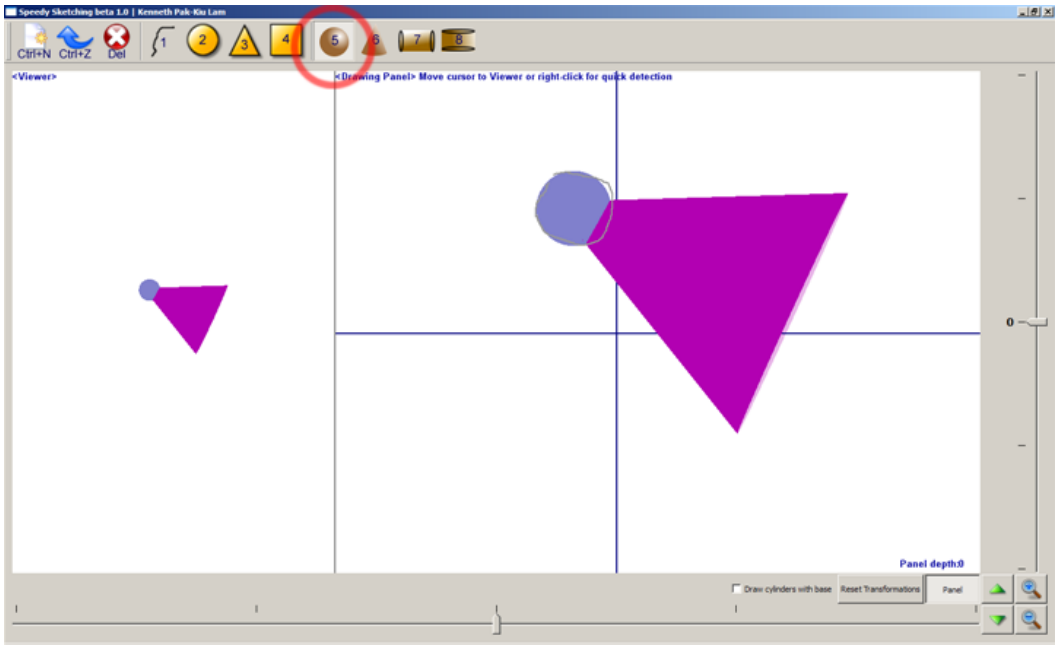


Figure 5.20: Embedding a sphere to a cone

In Figure 5.21, the user holds the right mouse button and drags the scene to the top-right corner, so there will be more space to sketch.

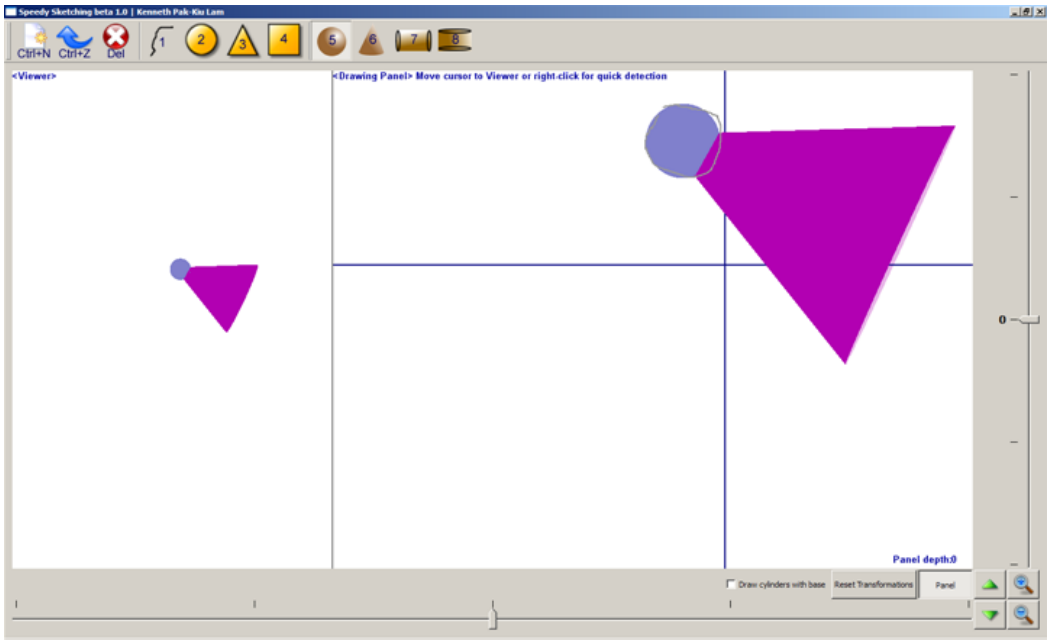


Figure 5.21: Dragging the scene

Then, the user proceeds to add 2 tubular cylinders and 2 spheres for the support of the lamp. Finally, they add a base to the lamp, which is formed by a drum shaped cylinder (long edges of the sketched rectangle are the circular edges).

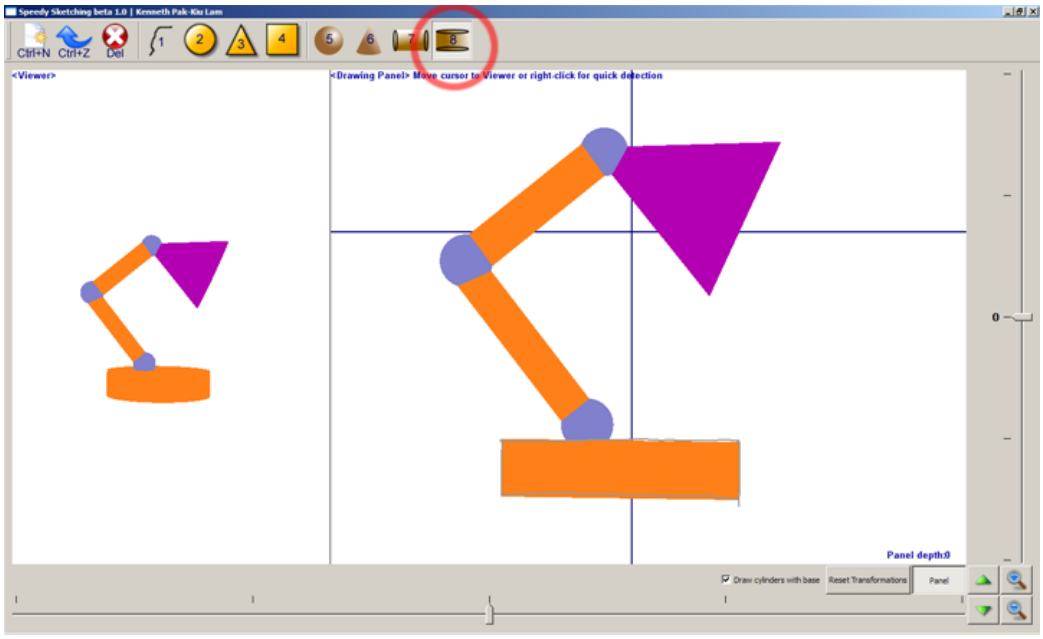


Figure 5.22: Adding tubular and drum shaped cylinders

The lamp is currently centred on the Drawing Panel, which bisects the lamp

vertically into 2 equal parts. Meanwhile, the user wants to add something to the side of the lamp base. The Panel is moved towards the user. As seen in Figure 5.23, the scene is automatically rotated to show the depth of the Drawing Panel. The Panel itself is transparent to show the bisection caused by it, which allows the user to determine when to stop sliding the Panel.

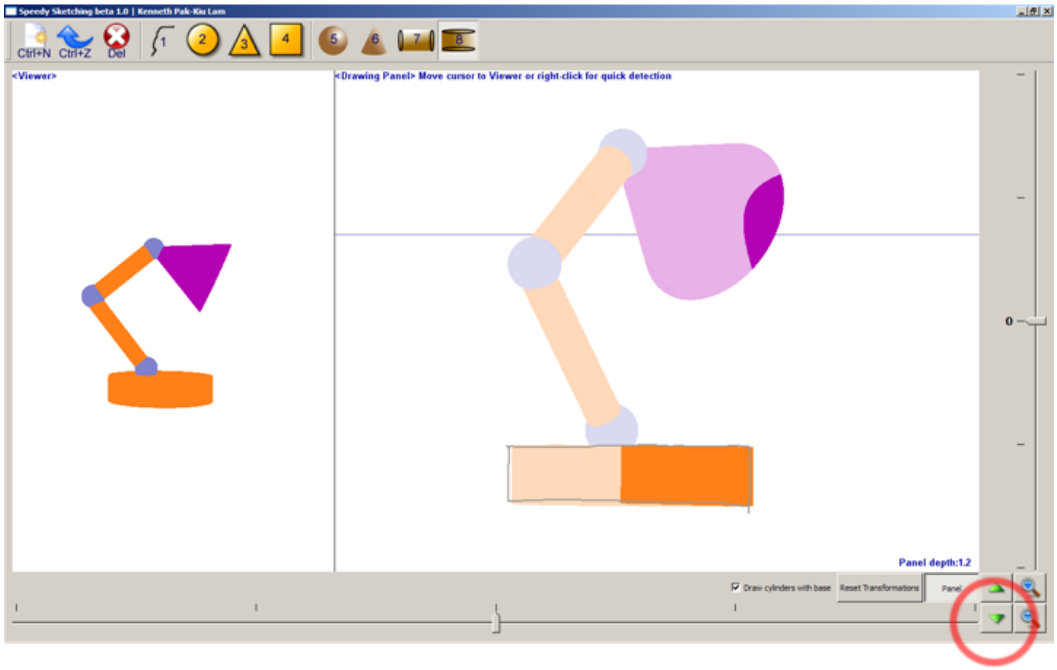


Figure 5.23: Moving the Drawing Panel depth

The user presses *zoom in* to show an enlarged view of the lamp. Selecting the tubular cylinder again, they draw the power button for a lamp (Figure 5.24).

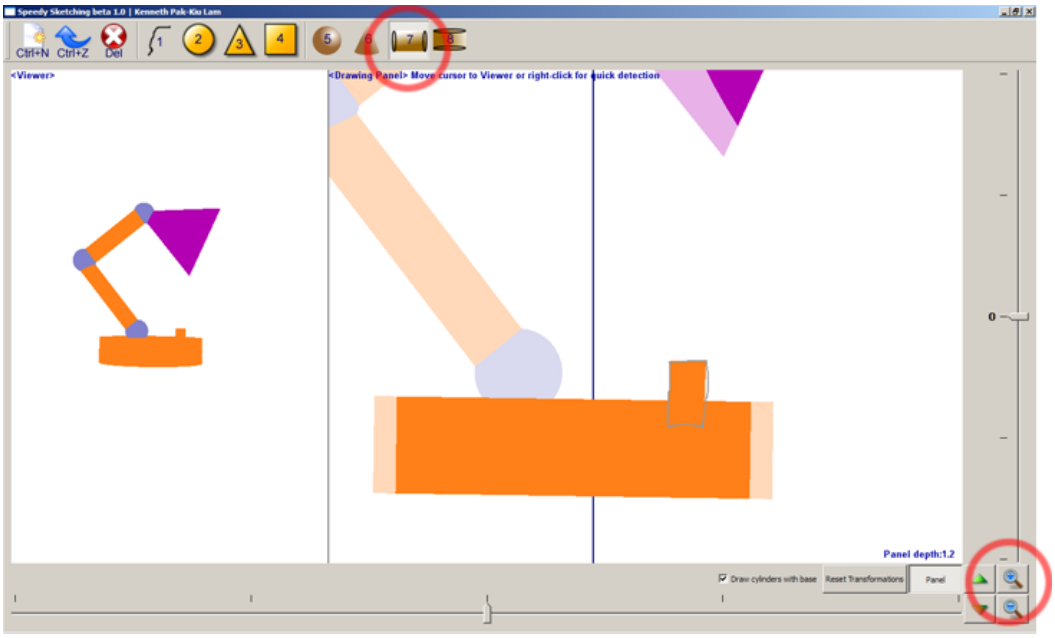


Figure 5.24: Zooming in to the scene

However, in Figure 5.25, the user is unhappy with the cone and deletes it with the *Delete* tool. The cone lights up to show that it is being selected.

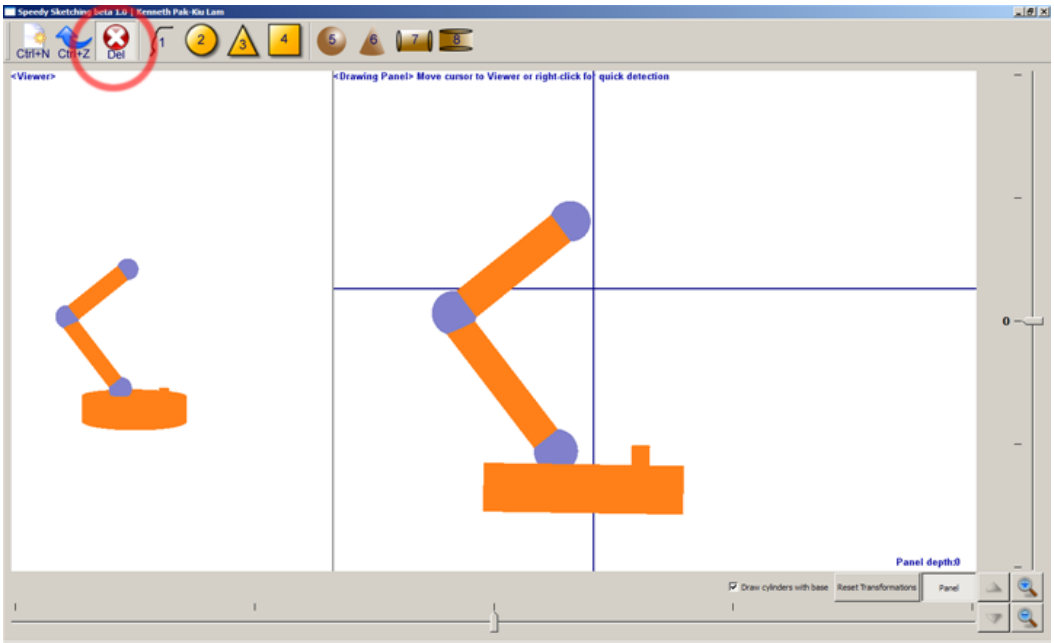


Figure 5.25: Using the delete tool

The user redraws the cone and also a sphere to represent the light bulb (Figure 5.26).

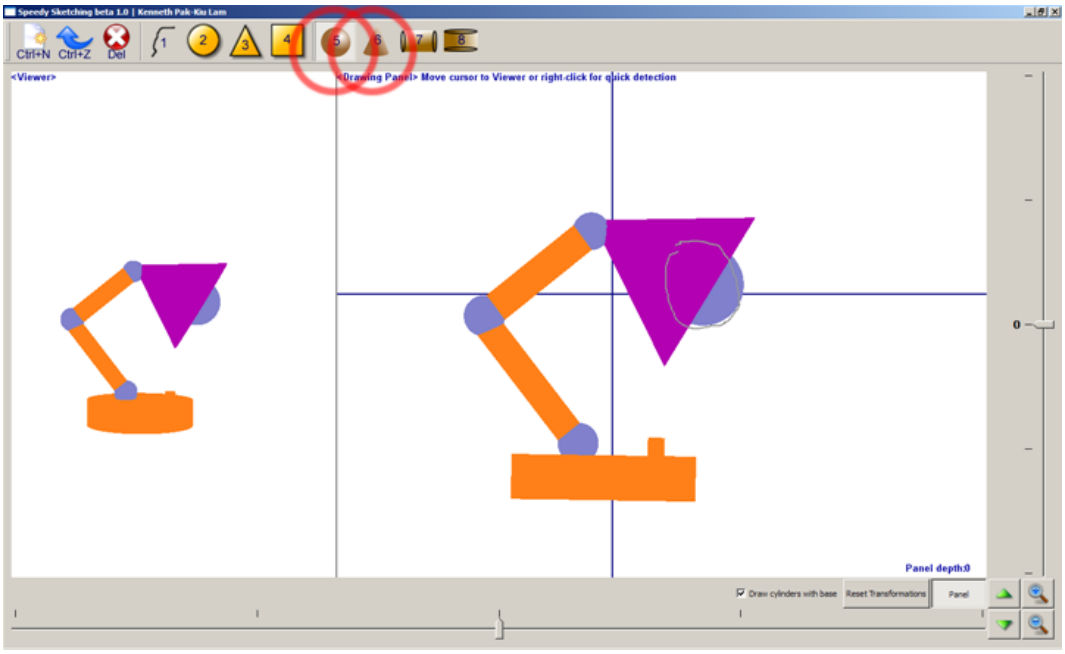


Figure 5.26: Redrawing shapes

Using the horizontal slider, the user rotates the scene along the vertical (Y) axis. As the Drawing Panel remains stationary, they draw a sphere which now appears on the right of the lamp (Figure 5.27).

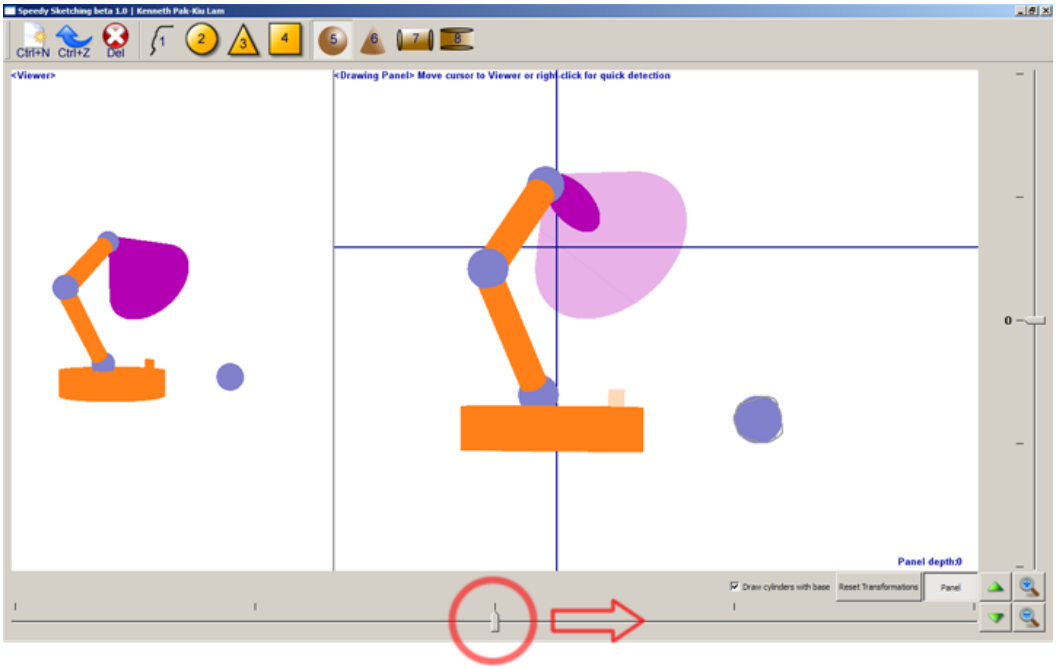


Figure 5.27: Rotating along the vertical axis

Using the same technique, 2 more spheres are drawn in different sizes and

distances from the lamp (Figure 5.28).

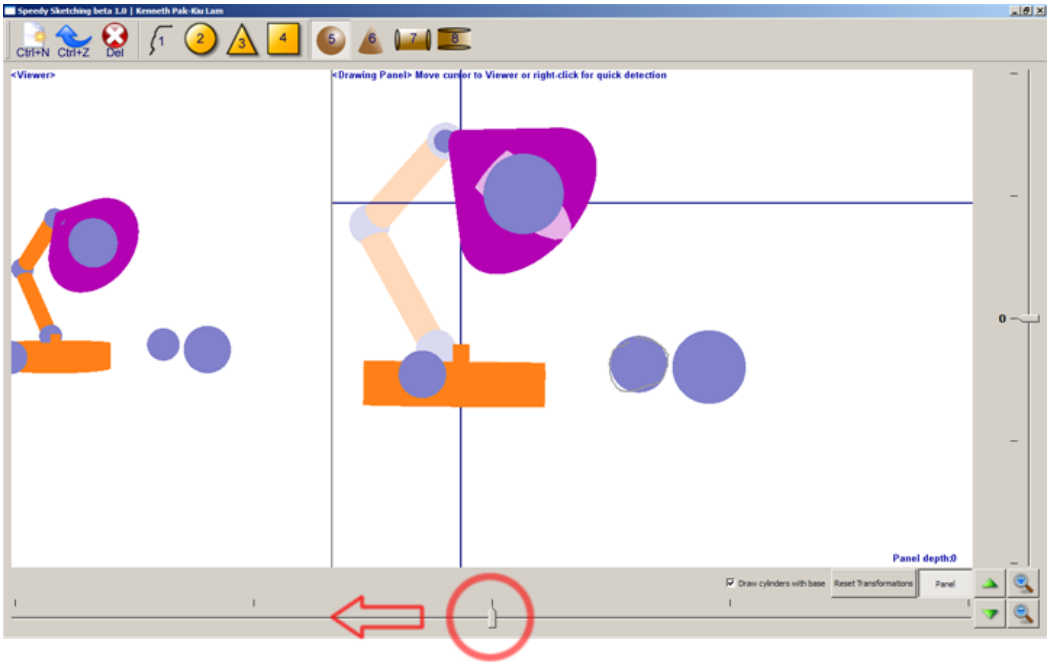


Figure 5.28: Rotating along the vertical axis (2)

Then, the user resets the scene with the *Reset Transformations* button. The lamp is again bisected into 2 equals halves by the Drawing Panel. The user unchecks *Draw cylinders with base* and draws 3 drum cylinders, which become rings.

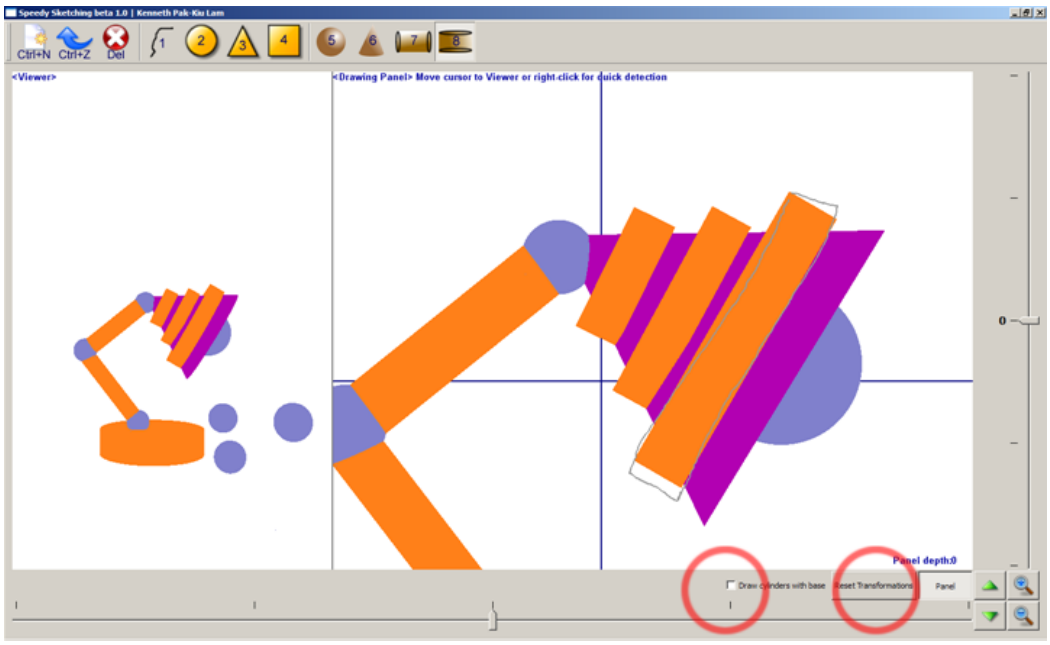


Figure 5.29: Resetting scene transformations and drawing cylinders with no bases

The user then rotates the scene in the horizontal (X) axis, and pushes the Drawing Panel away from themselves. A freehand line is drawn to represent the wire of the lamp (Figure 5.30).

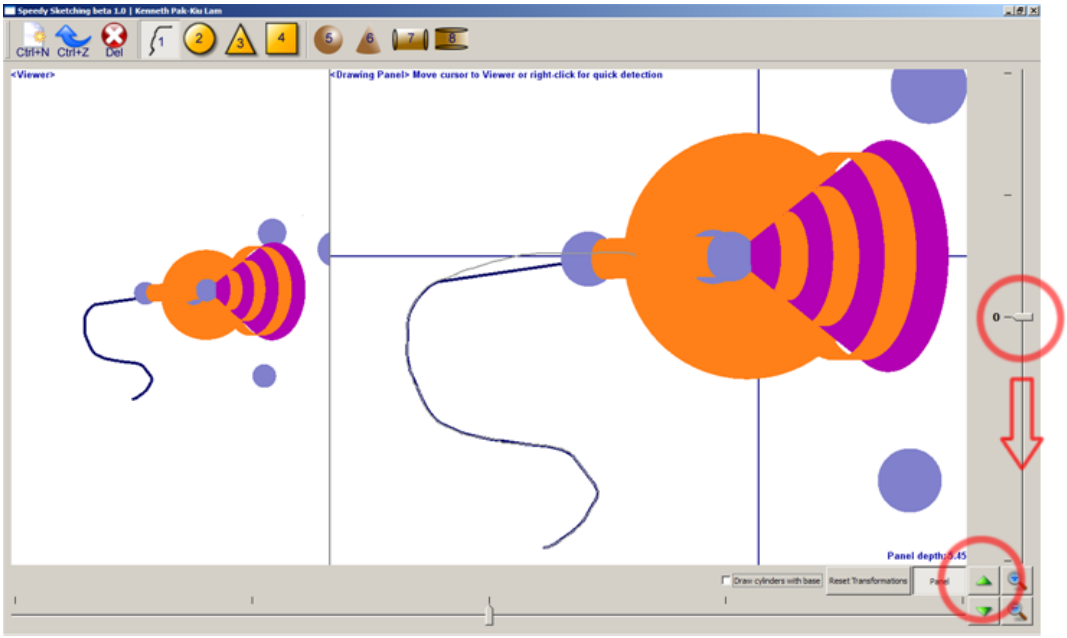


Figure 5.30: Rotating in the X axis

Figures 5.31 to 5.33 show the final result.

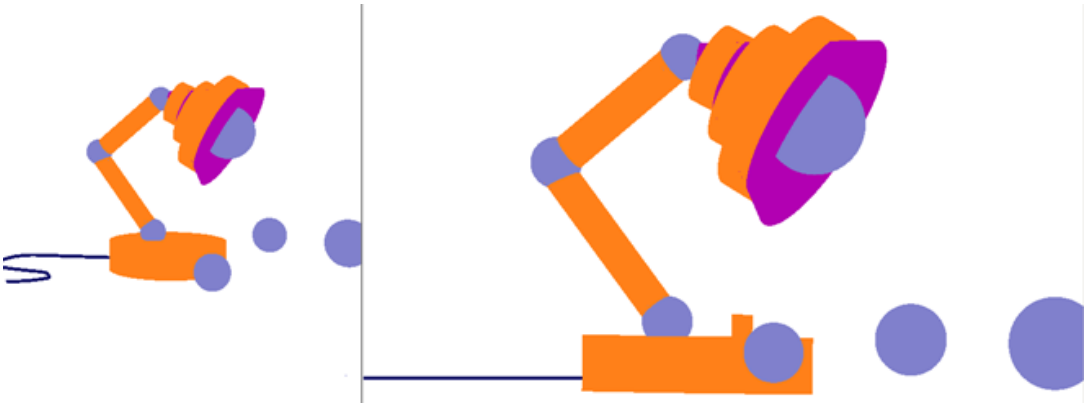


Figure 5.31: Screenshot of the scene (1)

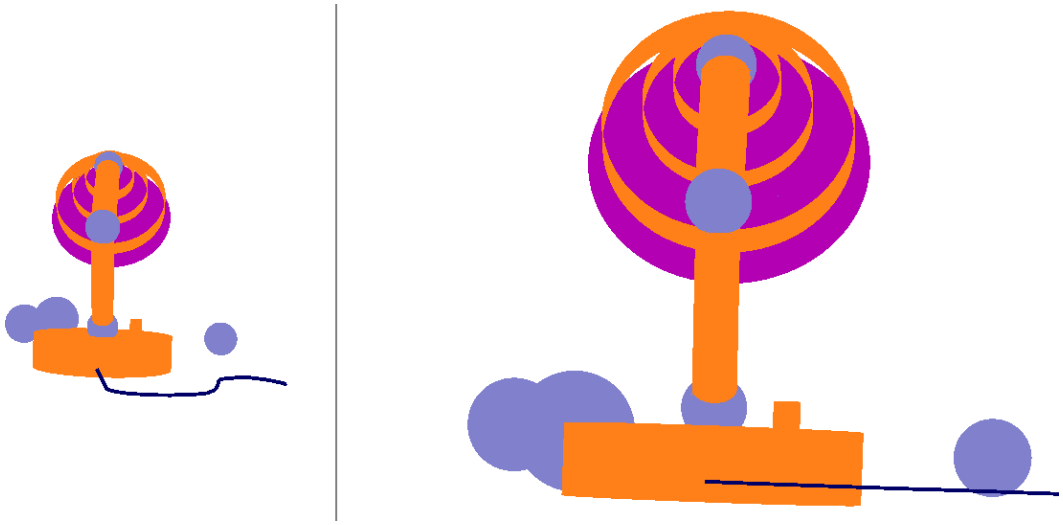


Figure 5.32: Screenshot of the scene (2)

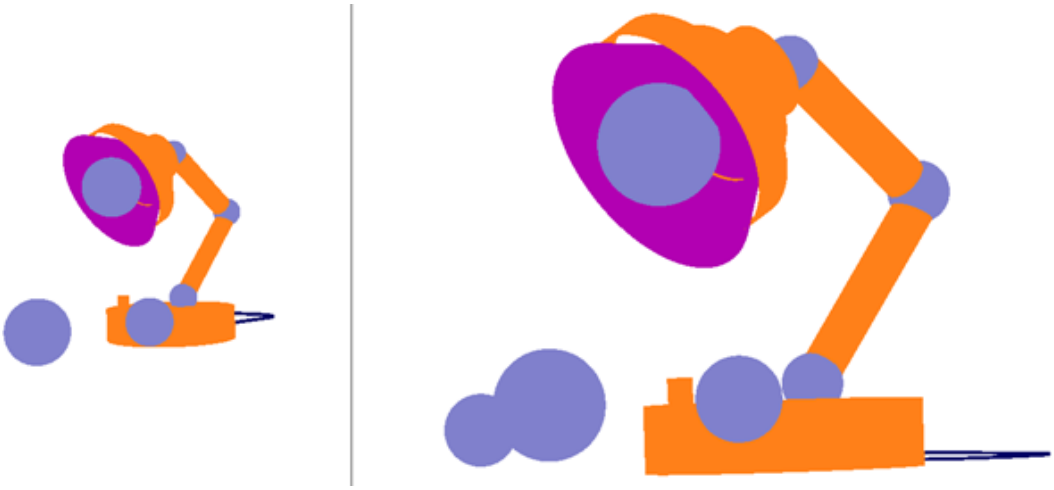


Figure 5.33: Screenshot of the scene (3)

5.6 Exporting Scenes

Due to limited time, saving and loading of scenes and exporting to .obj format have not been implemented. However, the author must stress that this is crucial for a useful system. At the moment, the user must rely on taking screenshots on their own for representing their results.

5.7 Performance

Much of the focus has been put in designing a simple user interface and algorithms for sketching intuitively. The performance of Speedy Sketching is deemed to be a secondary objective. However, some effort has been made to boost the performance of Speedy Sketching.

First of all, the freehand lines are on average reduced to between 42.9% and 66.2% of the original points. The reduction of memory complexity is also practised by all common shapes. This is achieved by only storing the parameters needed for generation of the shapes, instead of all point and face data. The ability of sketching common shapes means that display lists can be added in the future, as most shapes are generated by extremely similar OpenGL commands (the only differences being size and translations). Display lists allow caching of commands that are called frequently to boost performance. Some GPUs store them in dedicated memory or an optimized form to enhance compatibility [26].

When selecting objects to delete, it is found that the frame-rate is significantly lower. It is because the scene is drawn for determining the depth, then redrawn to be displayed every time the mouse moves. It is also discovered that when Speedy Sketching is just started up, it suffers from a short period in which the polling rate of the input interface is greatly reduced. On the other hand, the delay for detecting a shape is increased. While these may be caused by too many running programs, it will be good to investigate them for a more enjoyable user experience.

5.8 Known Issues

Albeit we strive to make Speedy Sketching bug-free, there are some known issues which have not yet been fixed. To make an all-round sketching tool which

sticks to the design specifications, attention will have to be paid in the following areas.

5.8.1 Rotation Sliders

According to the specification, the horizontal slider will rotate the scene along the Y axis and the vertical slider will rotate it along the X axis. The sliders should be independent of each other in any circumstance. However, due to the fact that the rotations are applied in a specific order and the objects are not at the origin, the rotation along the X-axis becomes dependent on the Y-axis rotation. When an X rotation is applied after a Y rotation, the X rotation does not revolve around the original X axis any more, but rather, a direction calculated from the Y rotation. This is illustrated in Figures 5.34 and 5.35 below.

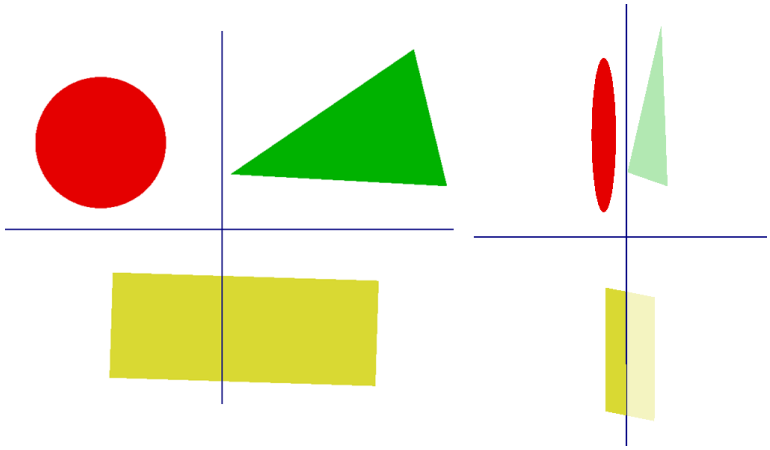


Figure 5.34: The original scene and after an 80-degree Y rotation

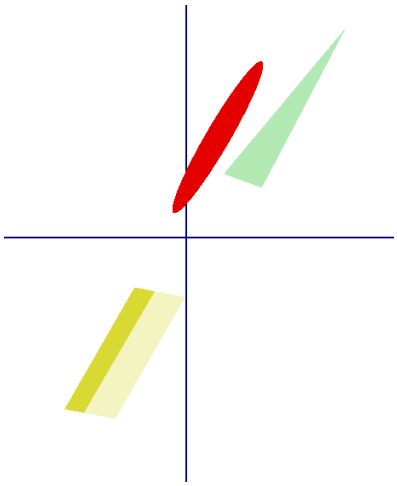


Figure 5.35: A subsequent X rotation reveals the scene does not rotate along the X-axis (horizontal line).

5.8.2 Drawing Panel Depth

The application rotates the view in the attempt of allowing the user to see the depth movement when sliding the Drawing Panel. However, this is not correctly implemented and sometimes the rotated view is slightly different from the original view. Where the rotated view does not show the bisection of an object, it could when the sliding has stopped (Figure 5.36). This would cause confusion and users may need to switch back and forth in order to see the correct depth.

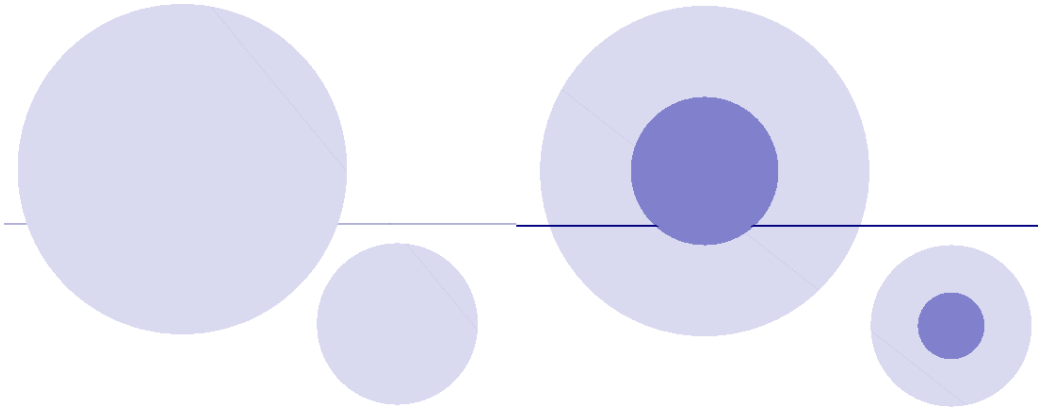


Figure 5.36: The rotated and the original view

5.9 Trivia: Drawing a Spring

Although the following was not expected during the design process, it is interesting enough to be included in this report. By adjusting the panel depth with

keyboard buttons (arrow up/down) while drawing a freehand line, it is possible to draw a line which moves around in 3D.

The user draws concentric circles continuously without releasing the button, while holding *arrow down* on their keyboard. This results in the change of the panel depth while drawing a line, and a spring is formed (Figures 5.37 & 5.38). While many other applications allow users to make Bezier curves in 2 different planes and combine them, we have, though unintentionally, provided an extremely easy way to draw curves in 3D. Despite the fact that it is very crude, and the users cannot visualize the current depth, this can be the basis of simplifying the process of making 3D curves.

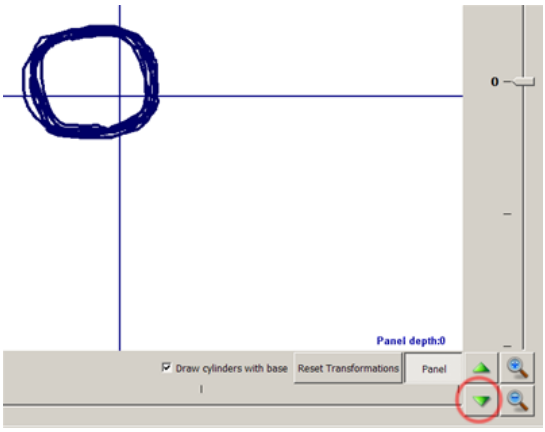


Figure 5.37: Front view of the spring

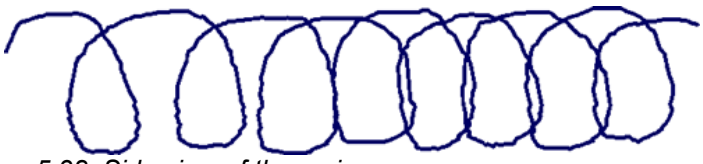


Figure 5.38: Side view of the spring

This chapter has provided in detail the results of applying implemented algorithms and techniques in Speedy Sketching. They are far from perfect and different problems exist. We hope that this will inspire fellow computer scientists to improve this project.

6 User Evaluation

6.1 Introduction

We have presented the results of the algorithms in the previous chapter, as well as some limitations of the application. However, sketching is an extremely subjective process, and users have varying expectations depending on their drawing habits. Therefore, it is impossible to evaluate Speedy Sketching mathematically. Also, we would like to prove the concepts of our interface design. For these reasons, a user evaluation has been carried out with 12 individuals, who have skill levels ranging from basic to expert. (Users are asked to rate their skill level among *basic, intermediate, advanced* and *expert*.) At the time of the evaluation, the extrusion of triangles and rectangles has not been implemented.

6.2 The Survey

Users are first required to familiarise with the interface. After that, they will be required to draw a 3D scene in Speedy Sketching, compare it with traditional software packages and comment on the improvements made. They will also go through the shape detection algorithms one by one and rate them. The questionnaire used has been included in Chapter 9.2 for reference.

As Speedy Sketching aims at creating a user interface so simple that no manual is required, we originally intended to provide no instructions to see how far we are from this goal. However, it is found that Drawing Panel depth, rectangle to cylinder conversion, triangle to cone conversion and activating the detection algorithms are all new ideas to the user, and it is hard to derive them from common sense. Therefore, instructions are added to the survey document.

6.2.1 Familiarisation and Overall Design

The users are asked to spend as much time as they want familiarising with the interface under no pressure. In this way, we can tell whether Speedy Sketching is simple and welcoming enough for new users. The results are extremely outstanding as the 12 users only need an average time of 6.83 minutes to familiarise with the interface. With such a little time needed for understanding the interface, we can be sure that we have achieved the goal of not intimidating new users. This objective is further supported by the first impressions of the users. All of them praised the intuitiveness, simplicity, clarity and the ease of use of the software.

Some users particularly like the icons provided, citing the large size for easy access and the distinctiveness between them. However, one user says he does not know that cylinder detection algorithm is expecting a rectangle input, despite we have explained it in the instructions. This shows that users are unlikely to pay great attention to instructions or a user manual, as they are not very visually or intellectually stimulating. This is why we have to spend even more time on interface design. Following the current trend of software, we can add an interactive tutorial in the application so first time users can go through all possible operations. Help can also be in the form of a virtual assistant which gives hints on whatever the user is doing. This is widely practised by the Microsoft Office suite, as users can learn on the go.

Two users say they do not understand the drawing panel depth movement. The author would like to agree that the operation is quite ambiguous, which a static picture can hardly explain. To improve this, a small example of moving the panel and drawing on 2 different planes can be displayed in the form of an animation. This can be triggered by an additional "*What is this?*" icon which explains the next

icon the user presses.

To our surprise, one user complains that there is no explanation of the Ctrl+Z (Undo) function, despite the icon looks similar to the ones used in popular applications. This proves that casual users are not likely to memorize icons and shortcuts. Therefore, we should explain all operations regardless of how often they are deployed in other programs.

6.2.2 Shape Detection

The users are then asked to rate all algorithms on a scale of 1 to 10, with 10 being flawless. Consistent with our test results, circle detection is the best rated algorithm with an average score of 9.0, followed by triangle detection which is rated 8.7. The only confusion for some of the users is sometimes the detection algorithm has to be activated manually (when the number of strokes is less than 3). As instructions for that are shown as a message on the Drawing Panel and explained in the survey, it is again proven that users are unlikely to read instructions, at least in the case of graphical design software. Action may be needed to attract the users' attention, by either having the message blink periodically on the screen or as a pop-up message when the application is started up.

The rectangle detection algorithm follows with an average score of 8.4. It is significantly worse rated than the triangle detection because, like our review results, users think it is less accurate. The score is also lowered because some users forget that only the 2-stroke algorithm has been implemented.

Freehand simplification and smoothing does not get a very good score – 8.0. Most users say they cannot tell the difference between the original and processed lines. This is actually a good result because it means the simplification does not alter the shape of the line too much while reducing the data size by up to 66.2%

(Chapter 5.3.5). However, to improve the algorithm, we may allow the users to adjust the number of iterations of McMaster smoothing and the angle for the angular tolerance algorithm. With these implemented, the users will be more likely to tell the difference and exploit the simplification to draw lines of different curvature and complexities.

The cone detection achieves an average score of 7.9. Many users discover that it is very difficult to get the cones to the orientation they want. Also, sometimes it is hard to draw a desired isosceles triangle. Being coherent with the review results, it is undoubtedly the weakest algorithm of all, and most work is required for improvement.

6.2.3 Deletion

All users think the delete operation is precise and clear, praising the highlighting of shapes for easy selection. We therefore think no further action is necessary.

6.2.4 Rotation and Panel Depth Movement

For the rotation sliders, we get mixed feedback when they are compared with the traditional trackball. Some users think that the split of rotation to 2 components will increase the time needed for modification. Others think it is better than the trackball; the separation will improve the perception of rotation for the users. Independent rotations will also prevent accidental rotations along an unintended axis. In the future, we can provide both options for users to choose.

Surprisingly, most of the users do not make much use of the Panel Depth, saying that it is not required for their sketches. In fact, we believe that our 2D to 3D conversions have played a part in the reduction of operations. Despite individual edges are not visible, the automatic conversion of, say, cylinder from

rectangle, puts edges around and on the Drawing panel without making the user worry about them.

6.2.5 Robustness

None of the users reported any crashes or unexpected behaviour during their testing processes. In addition to the reviews done by the author, we can safely say that Speedy Sketching is generally robust.

6.2.6 Comparison with Professional Software

Five of the twelve users have had some experience in professional 3D modelling packages like 3D Studio Max, SolidWorks and AutoCAD. It is interesting to note that 2 of them (both *advanced* users) gave up very quickly when using those packages and another say they are "too difficult for quick sketches". The fact that the self-proclaimed *advanced* users fail to learn the professional software highlights that it will be more than a mountain to climb for casual users. Subsequently, they all add that our application is the much easier alternative for quick sketches. We can tell that Speedy Sketching is superior in terms of user-friendliness, learning curve as well as simplicity, which are the basics of a well-designed interface.

6.2.7 Final Thoughts

Most of the users think Speedy Sketching is innovative in one way or another. Some say it is the rotation and depth movement, others say it is the shape generalisation algorithms which makes it groundbreaking. Some say it is the sheer simplicity which makes it stand out from others, and one user thinks the triangle to cone conversion is extraordinary. We believe that our objective of generating new ways of 3D sketching has been achieved.

However, an avid Photoshop user criticised it for being too simplistic, noting the

absence of grids, scales, markings and customisability. While this is true, the overuse of such things will mean we are making the same mistakes of some professional software. So, extreme care has to be taken when adding extra functionality. The separation of simple and advanced modes will be a good starting point for both users who want a simple interface, and advanced users who want more functionality.

One user is not happy with the lack of confirmation when *Close* is clicked, leading to possible loss of work. The author cannot help agreeing that the application is lacking in terms of standard features, e.g. saving, exporting screenshots, undo and redo. At the same time, we are delighted we have achieved an easy-to-use innovation. This is echoed by two of the users who think Speedy Sketching is a great tool for children to express their ideas in 3D or just be creative. The use of colourful shapes will also be aesthetically pleasing for young users.

6.3 Scenes Sketched by First Time Users

The following scenes are sketched by 4 of the 12 users, who have never used Speedy Sketching before.

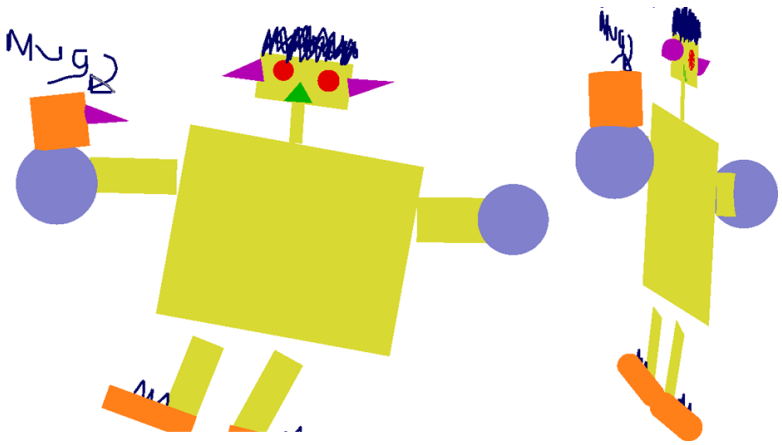


Figure 6.1: Scene by Peter Mcnerney

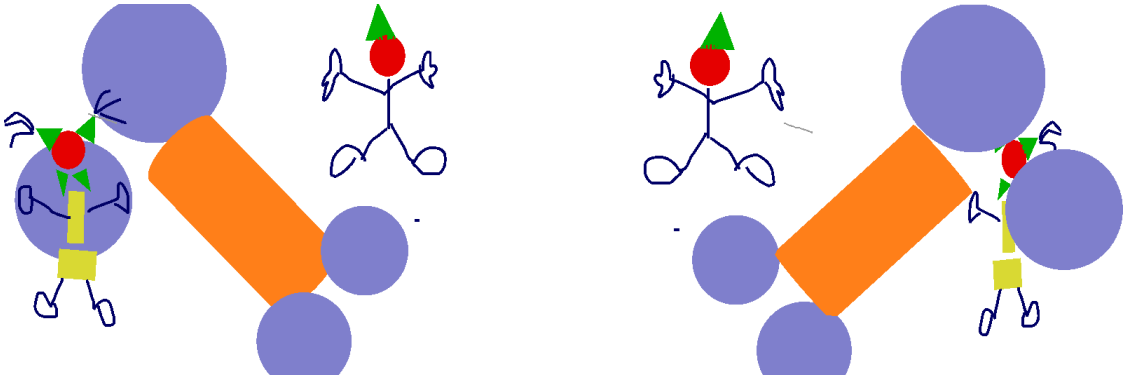


Figure 6.2: Scene by Lauriane Lancereau

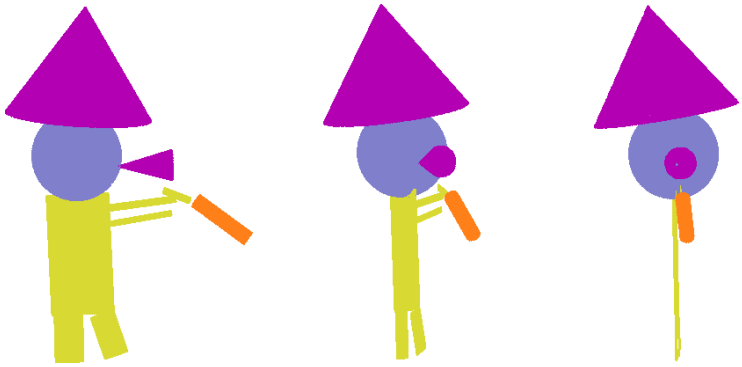


Figure 6.3: Scene by Viridis Liew

Figure 6.4: Scene by Samantha Bail

From the user evaluation, we have obtained valuable insight from a user's perspective, which can be used to expand and improve Speedy Sketching. Understanding the users' needs has always been a critical aspect in software

development and it is no different for our research. One of the major findings of this evaluation is, perhaps not surprisingly, the fact that users do not tend to read manuals. It is our responsibility to adapt accordingly so the application is properly used.

7 Conclusion

7.1 Speedy Sketching

Why should 3D modelling be any different from 2D sketching? The addition of one extra dimension puts casual users off, and it is surprising because graphical user interfaces have been around for decades. Apparently, more work needs to be done on the interface of 3D modelling software so it will no longer be viewed as a "professional" utility. The lack of simple 3D software is limiting creative ideas to the flat and lifeless realm of 2D, and it can be time consuming to give them life – convert them to 3D. Designers should be able to express their ideas in 3D vividly in the first place.

Speedy Sketching comes in to allow users to sketch simple 3D shapes which everyone is familiar with. The sketching process is made simple – sketching the 2D silhouette of any 3D shape feels intuitive – because it is human nature; it is something we all used to do as a child. The fact that all shapes are automatically generalised means human errors are dealt with, and we have an electronic sketching system in which presentation is enhanced but naturalness is maintained. The drawing method is consistent for all shapes, thus greatly reducing the time needed to learn the software.

We have designed the interface with simplicity in mind. There is no traditional drop-down menu which is time consuming to traverse. At the same time, all operations are activated by one key stroke. The main drawing area is large and clear and generated with an orthogonal view to allow alignment of shapes at different depths. This is supplemented by a perspective view which shows the

scene in a more realistic mode. Apart from translating the scene with the right mouse button, all other operations are done by the left mouse button, so there is little to remember for the users. The shortcuts are embedded in the icons themselves, removing any need for casual users to memorize them. The user evaluation shows that most users are happy with the easy-to-use, intuitive and simple interface. We have set ourselves apart from the intimidating professional modelling software. In contrary, Speedy Sketching is welcoming. It reinforces that it is the software engineer's job to understand the users, and not the other way round.

Making use of simple 2D geometry equations, we have developed several shape detection algorithms: circle, triangle and rectangle. They are exploited to make shapes in 3D: sphere, cylinder, cone, rectangular and triangular prisms. While we are extremely delighted with the circle and triangle detection algorithms, work is needed to improve the determination of cone orientation and the inaccuracies in rectangle detection. The addition of detection of intersection between strokes will allow a wider range of user behaviour, further improving user experience.

It should be stressed that Speedy Sketching is not meant to be a fully-operational commercial software. The main objective of this research is to achieve something new which we have done. Consequently, some standard procedures are left out, for example, the loading and saving of Speedy Sketching scenes, the exporting of scenes in .obj format and a colour chooser. It is hoped that the application will be expanded by enthusiasts who will continue to build a full-fledged 3D modelling software.

All in all, given the implementation of an extremely simple and intuitive interface, and the invention of 2D silhouette to 3D conversion, we believe this research is a success. Speedy Sketching is one of the efforts towards creating a

perfect 3D modelling software, and we are proud to be a part of an upcoming revolution.

7.2 Future Work

Although we reckon that Speedy Sketching is a brilliant innovation, it is nothing more than a work-in-progress. Much effort is needed in many areas before it can be a useful application.

Regarding the interface where shortcuts are provided for quick selection of shape detection algorithms, we have only taken a United States keyboard into account, and failed to notice that some keyboards have different layout of buttons. For example, numeral and symbol buttons are flipped on a French keyboard, and *Shifting* is required to activate the numerals. This means 2 buttons have to be pressed even for changing the mode, which destroys the original purpose of quickness. Also, it is relatively harder to press 2 buttons with one hand if the user is using a mouse with another. To make a better user interface, the application will not only need to understand the user's actions, but also, the user's environment. It may be required to detect the type of keyboard the user is using and change the icons and shortcuts automatically. Additionally, we can allow the users to customise the shortcuts according to their own habits.

Secondly, it is impossible to see if objects of the same type are overlapping in the scene. Only with delete mode, we can highlight the objects one by one and examine if redundant objects are present. In Figure 7.1, the delete function reveals 2 hidden triangles that cannot be seen otherwise. This causes much confusion. The user may also do this intentionally only to find that the desired effect cannot be achieved, as all triangles have the same colour. To solve this, we can draw lines to represent edges of a shape. Apart from being helpful for

overlapping shapes, it will be perfect for the extrusion operation as well. At the moment, it is extremely difficult to tell the orientation of a rectangular prism in the orthogonal view, because no edges are displayed (Figure 7.2). It may also be helpful to have a mode where lighting is applied, so the presence of shading can help the user identify 3D shapes – especially for spheres, cones and cylinders in which edges can hardly be added.

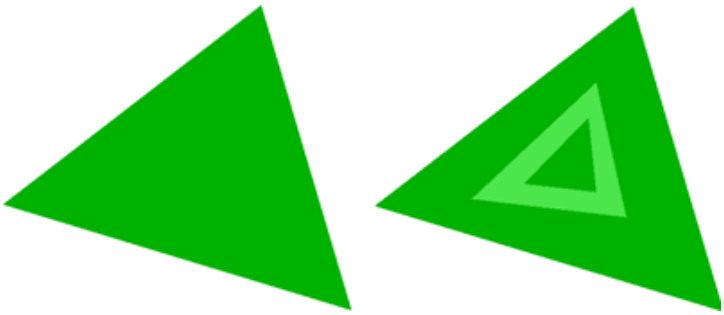


Figure 7.1: Two triangles are hidden in a larger one

Figure 7.2: Rectangular prism (front view, 45-degree and 90-degree rotations applied)

The reviews in Chapter 5 revealed that the shape detection can be improved by adding more intelligence. Currently, we limit each stroke to one or the multiple of a side, which is hardly the case when we draw. Also, intersections between strokes are ignored while some of the users use them to define vertices. This highlights the fact that more flexibility is needed in a sketching system, because users have different habits and behaviour. The only way for the system to learn about this kind of information is through training or customisation. One way of making the system learn on-the-go would be modifying it to become a suggestive interface, where the user is made to choose the shape that best describes their sketch.

Through collecting data on the user's choices, their behaviour can be learnt and the system can make more accurate predictions.

From the equilateral triangle problem mentioned in Section 5.3.3, it is found that even an intuitive sketching interface can suffer from ambiguities. Although graphics can normally contain a lot of information, illusions and other geometric properties may cause the system to misunderstand the user. This is why supplementary information is needed. Where such ambiguities exist, the system must be intelligent enough to request for more information in order to produce the right results. It is important to note that requesting for such information will cause a delay in detection, so some users may prefer providing it during sketching. Such option should be available for advanced users who are keen to improve construction speed.

Originally, it is feared that with too many options for the users to adjust, Speedy Sketching may become one of the intimidating professional software. However, it is undeniable that some users do need more flexibility in the application to obtain the desired results. The presence of rulers, line thickness adjustments and grids would not confuse the casual user if they can be hidden. The easy solution would be the introduction of a simple and advanced mode. While simple mode may be appealing to casual users, the advanced mode can help the keen users make precise and professional looking models. Mode separation is widely practised in anti-virus software which we can learn from.

One of the findings of this research is the reluctance of users to read the manual. It is inevitable because the sketch interface is extremely interactive but the manual is instead static, hence it is much less stimulating. However, this does not mean that nothing can be done. By adding tutorials and hints to the interactive interface itself, the need for a text-based manual can be nullified. This

will also mean that users will not have to switch between the help file and the application, which is troublesome. The addition of an interactive tutorial will enhance the understanding of the workings of the software. Users will be less likely to be intimidated by new concepts being able to test them at the very moment they learn.

We believe that if all of the above is achieved, it will make the basis of a successful sketching interface. We sincerely hope that someday, 3D modelling will finally be as easy as sketching with a pencil and a piece of paper.

8 References

[1] Launday, J. A., University of California, Myers, B. A., Carnegie Mellon University (2001). *IEEE Computer*, vol 34, no. 3 (March 2001), pp 56-64

[2] *Excerpts from A Conversation with Gordon Moore: Moore's Law* (Video Transcript), Intel Corporation (2005). From ftp://download.intel.com/museum/Moores_Law/Video-Transcripts/Excepts_A_Conversation_with_Gordon_Moore.pdf Accessed April 11, 2009 16:45

[3] Gross, M. D. and Do, E. Y. (1996). *Ambiguous Intentions: A Paper-like Interface for Creative Design,* Proc. ACM Symp. User Interface Software and Technology, ACM Press, New York, pp. 183-192. Cited in [1]

[4] Goel, V. (1995). *Sketches of Thought,* MIT Press, Cambridge, Mass. Cited in [1]

[5] LaViola, J. J. (n.d.). *MathPad²: A System for the Creation and Exploration of Mathematical Sketches,* from http://www.cs.brown.edu/~jjl/mathpad/ Accessed April 12, 2009 15:30

[6] LaViola, J. J. (2007). University of Florida "*An Initial Evaluation of MathPad²: A Tool for Creating Dynamic Mathematical Illustrations", Computers & Graphics: An International Journal of Systems & Applications in Computer Graphics: Algorithms & Techniques for Interaction, Multimedia, Modelling and Visualization,* Elsevier Ltd., vol 31, Issue 4 (Aug 2007), pp 540-553

[7] Buttenfield, B. P. and McMaster, R. B. (1991). *Map Generalization: Making rules for knowledge representation,* Longman Group UK Limited, pp 3-10

[8] Jenks, G. F. (1981). *Lines, Computers and Human Frailities,* Annals of the Association of American Geographers, vol 71(1), pp 1-10. Cited in [9]

[9] McMaster, R. B., and Shea, K.S. (1992). *Generalization in Digital Cartography,* Association of American Geographers, pp 27-58 Cited in [16]

[10] Igarashi, T., Matsuoka, S., Tanaka, H. (1999). *Teddy: A Sketching Interface*

for 3D Freeform Design, ACM Siggraph, University of Tokyo, Tokyo Institute of Technology

[11] Igarashi, T. (2003). *SmoothTeddy: Quick 3D Modelling and Painting*, from http://www-ui.is.s.u-tokyo.ac.jp/~takeo/java/smoothteddy/index.html Accessed April 14, 2009 17:30

[12] Igarashi, T., Hughes, J.F. (2003). *Smooth Meshes for Sketch-based Freeform Modelling,* University of Tokyo and Brown University, pp 1-2

[13] Masry, M., Kang, D., Lipson, H. (2005). *A freehand sketching interface for progressive construction of 3D objects*, *Computers & Graphics: An International Journal of Systems & Applications in Computer Graphics: Algorithms & Techniques for Interaction, Multimedia, Modelling and Visualization*, Elsevier Ltd., vol 29, Issue 4 (Aug 2005), pp 563-575

[14] Lipson, H. (n.d.). *Cornell CCSL - Research - 3D Sketching,* from http://216.139.212.17/mae/ccsl/research/sketch/index.html Accessed April 20, 2009 20:00

[15] Google Inc. (2009). *SketchUp Blog,* from http://sketchupdate.blogspot.com/search/label/User%20Stories Accessed April 21, 2009 18:20

[16] Ellis, F. (n.d.). *Line Generalisation Algorithms - Lang Visualisation,* from http://www.sli.unimelb.edu.au/gisweb/LGmodule/LGMcMastersVisualisation.htm, University of Melbourne, Accessed March 23, 2009 13:47

[17] McReynolds T. and Blythe D. (2005). *Advanced Graphics Programming Using OpenGL,* Elsevier Ltd, Preface, pp xxiv

[18] Nokia (n.d.), *Qt - A cross-platform application and UI framework,* from *http://www.qtsoftware.com/products/library/modular-class-library#advanced-3d-graphics-opengl* Accessed April 23, 2009 23:10

[19] Erleben, K., Sporring, J., Henriksen, K., Dohlmann, H. (2005). *Physics-Based Animation*, Charles River Media, Inc.

[20] OpenGL Architecture Review Board (n.d.). *OpenGL Programming Guide: The Official Guide to Learning OpenGL (The Red Book) v1.1,* from http://glprogramming.com/red/chapter11.html Ch. 11. Accessed April 25, 2009 21:45

[21] Howard T. (2009). *An Introduction to Graphics Programming with OpenGL:*

Tutorial and Reference Manual, University of Manchester, pp 41

[22] Wikipedia (n.d.). *WYSIWYG*, from http://en.wikipedia.org/wiki/WYSIWYG, Accessed Aug 15, 2009 22:47

[23] Wolfram Math World (2003). *Point-Line Distance – 2 Dimensional*, from http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html Accessed Jul 29, 2009 1:00

[24] Integrated Publishing (n.d.). *Angle between 2 lines*, from http://www.tpub.com/math2/5.htm Accessed Jul 27, 2009

[25] Edward Angel (2006). *Interactive Computer Graphics – A Top-Down Approach using OpenGL.,* Pearson Education Inc., Ch. 3, pp 124-131

[26] OpenGL Architecture Review Board (n.d.). *OpenGL Programming Guide: The Official Guide to Learning OpenGL (The Red Book) v1.1,* from http://glprogramming.com/red/chapter07.html Ch. 7. Accessed Aug 29, 2009 14:20

[27] Preece J., Sharp H., Rogers Y. (2007). *Interaction Design: beyond human-computer interaction,* John Wiley & Sons, Ltd. Ch.1, pp 2-6

[28] Two Forty-eight AM: The B sides (2006). *Review: Harmony 880 Remote,* from http://www.248am.com/mark/reviews/review-harmony-880-remote/ Accessed Aug 30, 2009 16:30

9 Appendix

9.1 User Manual

9.1.1 Modes of Operation

Figure 9.1: The menu bar

The Menu Bar consists of 12 icons. The first 2 are spontaneous action icons, *New (Ctrl+N)* and *Undo (Ctrl+Z),* which clears the scene and removes the last shape drawn respectively.

The remaining 10 buttons are different modes of operation, whose states persist until the mode is changed. From left to right, they are *Delete, Freehand Draw, Circle, Triangle, Rectangle, Sphere, Cone, Tubular Cylinder, Drum Cylinder and Extrusion.*

You can change the mode of operation at any time.

Other controls include the following:

Reset Transformations: Resets the zoom, rotation and Drawing Panel depth to the default levels

Draw cylinders with base: Uncheck if you do not want a top and base for your cylinders

Panel: The panel can be disabled to show the object in clear view

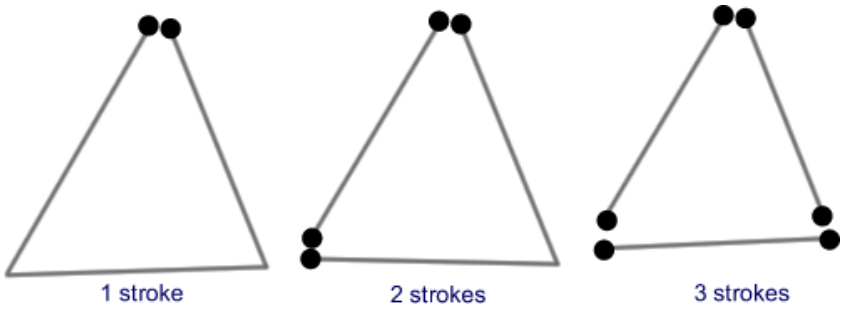
9.1.2 The Shapes

Speedy Sketching detects 8 different kinds of shapes.

Freehand: Draw any lines in any way you want

Circle: Draw a circle

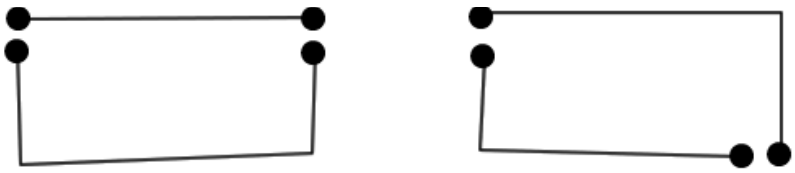
Triangle: Draw a triangle with 1-3 strokes, as below. If it is less than 3 strokes, you will need to activate the detection algorithms by *Right-click* or moving your cursor to the View Panel.

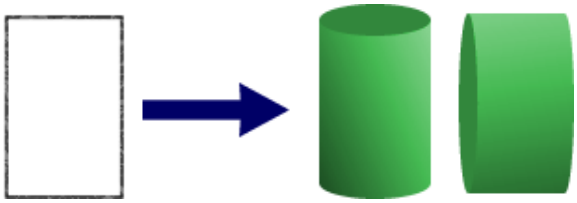


Sphere: Draw a circle.

Cone: Draw an isosceles triangle. Like a normal triangle, you have to activate the detection if drawn with 1-2 strokes.

Rectangle and Cylinders: Draw a rectangle with 2 strokes, as below:





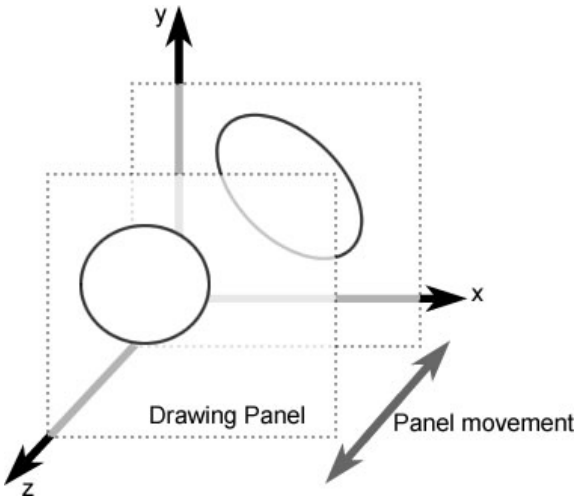
Tubular cylinder (left): A tubular cylinder refers to the rectangle where the long edge is the height of the cylinder and the short edge the circular bases.

Drum cylinder (right): A drum cylinder is opposite to the tubular cylinder. The long edge of the rectangle is the circular base and the short edge is the height.

9.1.3 3D Concepts

The horizontal slider rotates the scene along the vertical line on the Drawing Panel, and vice versa. Regardless of the rotation, the Drawing Panel will always be parallel to the screen.

The green arrow buttons slides the Panel towards and away from the camera.



Up – Away from camera, Down – Towards camera

9.1.4 Left Mouse Button

The left mouse button is used for most operations on the screen.

Sketching a shape: When in drawing mode, press and hold the button to draw

lines.

Extrusion of a shape: Drag towards the right to increase the height, and the left to decrease it.

Deletion of a shape: When in delete mode, highlight the shape you want to delete, and click once.

9.1.5 Right Mouse Button

The right mouse button can be used to move the scene in the plane parallel to the screen.

9.1.6 Shortcuts

Speedy Sketching provides keyboard shortcuts for quick operations.

Ctrl+N: clears the scene

Ctrl+Z: Undoes the last shape drawn

Del: Switch to delete mode

1-8: Shape detection algorithms

0: Extrusion mode

Arrow Up: Move drawing panel away from the camera

Arrow Down: Move drawing panel towards the camera

+: Zoom in

-: Zoom out

Alt+F4: Quit the application

9.2 Survey used for User Evaluation

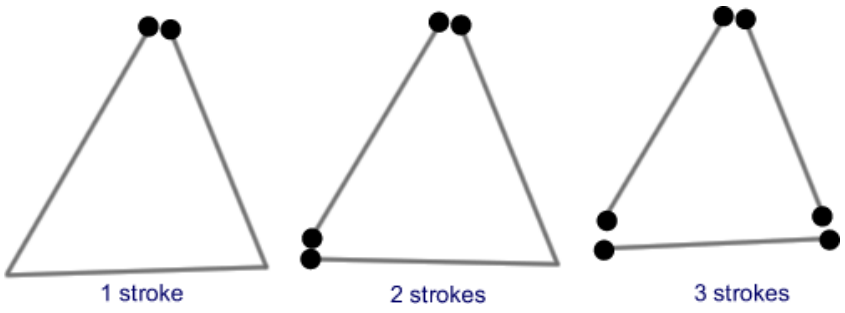
Speedy Sketching User Survey

Kenneth Pak-Kiu Lam | keniflam_at_hotmail_dot_com

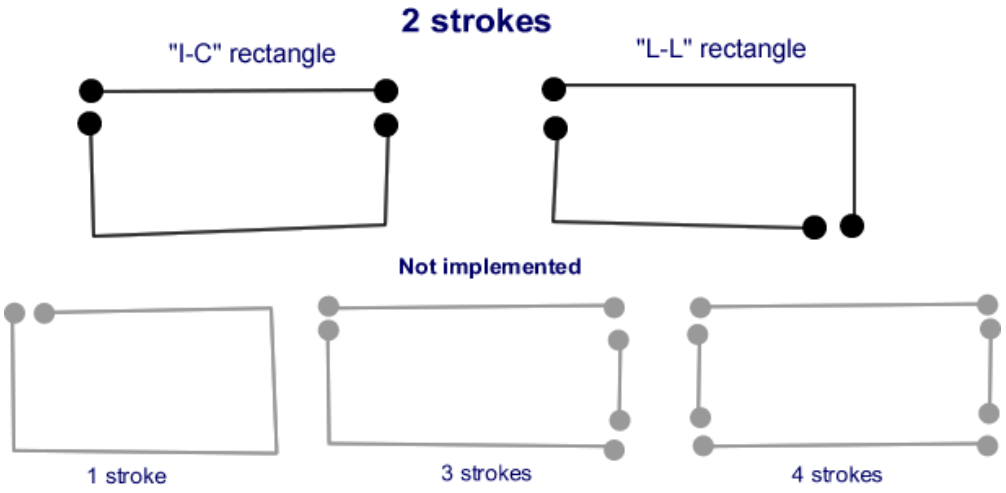
Supervised by Toby Howard

This application aims at providing a simple interface for sketching 2D and 3D models quickly. The first step requires you to run the application and familiarise yourself with the interface, until you believe you know how everything works or you are stuck. Please read the following notes before you start.

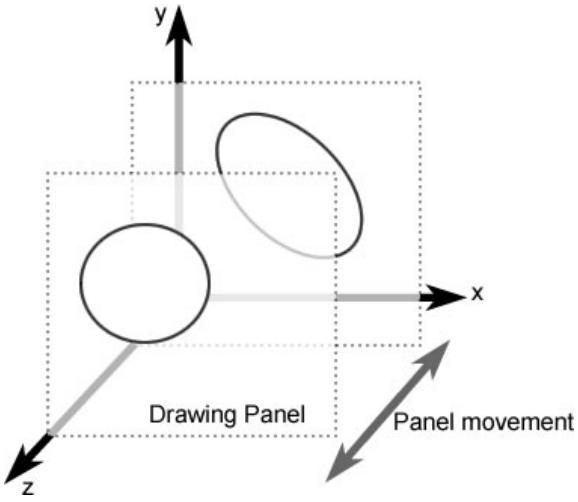
NOTE 1: You can draw a triangle with 1 to 3 strokes, like the following:



NOTE 2: because of time constraints, only a 2-stroke rectangle can be detected (as shown below). ***Rectangle*** *and* ***cylinder*** *detection algorithms both require you to draw a rectangle.*



NOTE 3: Activate the detection algorithms by moving your cursor to the View Panel or right-clicking



*NOTE 4: The **arrow buttons** move the drawing panel towards and away from the user.*

<u>Part 1: Familiarisation</u>

How much time did you take to familiarise with the interface?

Ans:

Did you find any operations that you don't understand even after your initial familiarisation? What icon / button is it?

Ans:

How do you find the overall design of the interface?

Ans:

Part 2: Algorithms

Now, press Ctrl+N or the top left icon to clear the scene. You are now required to draw each shape and rate the shape detection algorithms. To explain your comments, you can post screenshots into this document. To make a screenshot on Windows, press (Alt+Prnt Scrn) and then (Ctrl+V) to paste it.

Firstly, the freehand algorithm. It simplifies the line automatically.

On a scale of 1 (worst) – 10 (best), rate the simplification algorithm. Explain why if the score is lower than 9.

Ans:

On a scale of 1 (worst) – 10 (best), rate the *circle/sphere* detection algorithm. Explain why if the score is lower than 9.

Ans:

On a scale of 1 (worst) – 10 (best), rate the *triangle* detection algorithm. Explain why if the score is lower than 9.

Ans:

On a scale of 1 (worst) – 10 (best), rate the *cone* detection algorithm (draw an isosceles triangle). Explain why if the score is lower than 9.

Ans:

On a scale of 1 (worst) – 10 (best), rate the *rectangle/cylinder* detection algorithm. Explain why if the score is lower than 9.

Ans:

Now, using the delete function, select and delete any shapes you have drawn. How did you find the operation?

Ans:

Part 3: Rotation, Draw Panel movement

What do you think about the rotation sliders? Is it an improvement over the traditional virtual trackball and why?

Ans:

What do you think about the panel depth movement? (arrow up/down)

Ans:

Part 4: Scene drawing

Now spend 5 minutes or more to draw anything you want, and paste a few screenshots (with different angles) here. Your work may appear in the gallery section of the dissertation so readers can get a better idea of what the application can do.

Briefly describe how you feel about the usability and simplicity of Speedy Sketching.

Ans:

Part 5: Final thoughts

Has the application crashed / behaved irregularly during this survey? If so, please explain what you were doing:

Ans:

What 3D model design packages have you used before? Briefly compare Speedy Sketching and the professional package you used in terms of user friendliness, learning curve and speed.

Ans:

What do you think is the biggest innovation of Speedy Sketching?

Ans:

Write here if you have any final comments:

Ans:

Personal information

Finally, please fill in the following personal information. To thank you for your participation, your name will appear in the "Acknowledgements" section of the dissertation, but you can remain anonymous. You details will only be used for the purpose of this dissertation, Speedy Sketching. None of your personal information will be published.

Full name:

Job/Expertise:

Age:

Contact e-mail:

Years of experience in using a computer:

Rate your computing skills (basic/intermediate/advanced/expert):

Please send this document to keniflam_at_hotmail_dot_com

Thank you very much for your time! - Ken

9.3 Gallery of Speedy Sketching Scenes

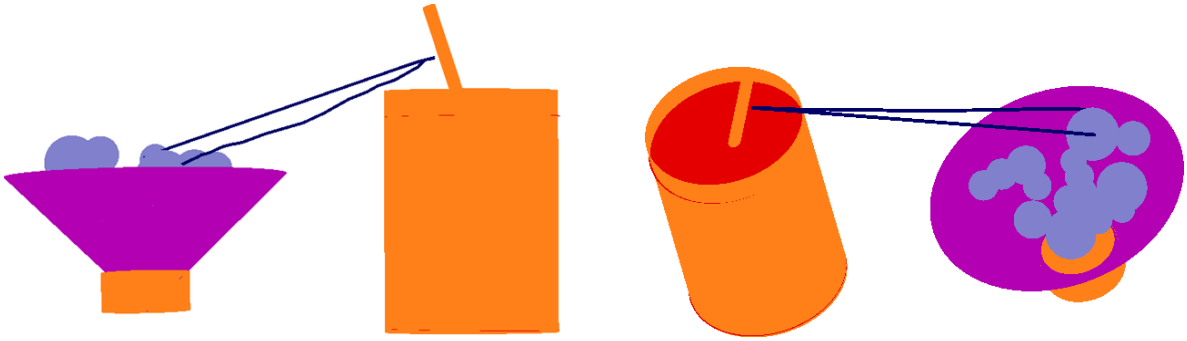


Figure 9.2: A bowl and a can of soft drink

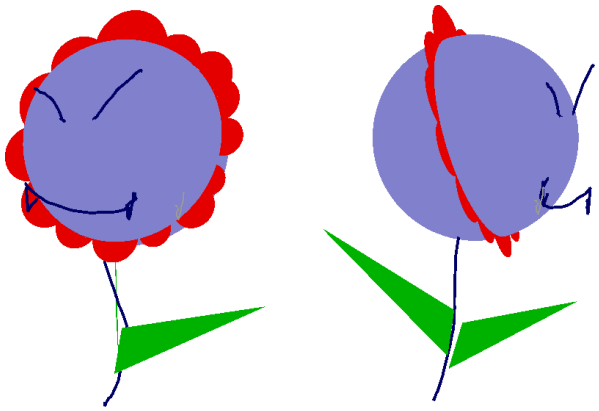


Figure 9.3: An "evil flower"

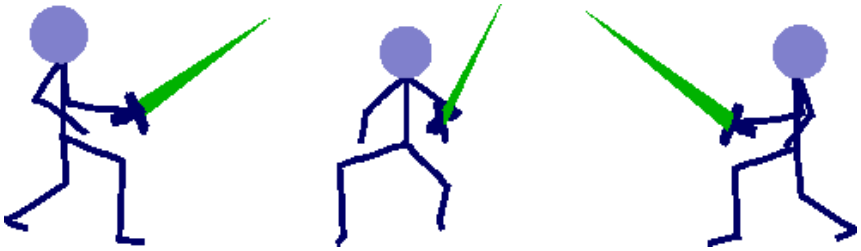


Figure 9.4: A sword-wielding person

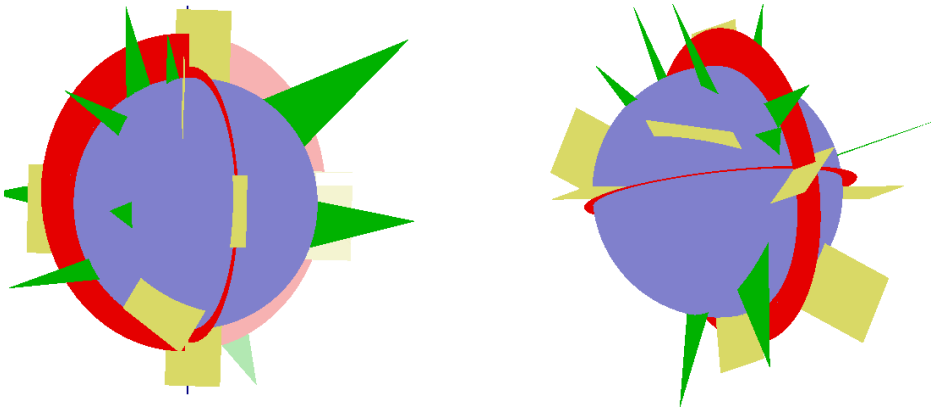


Figure 9.5: A decorative sphere

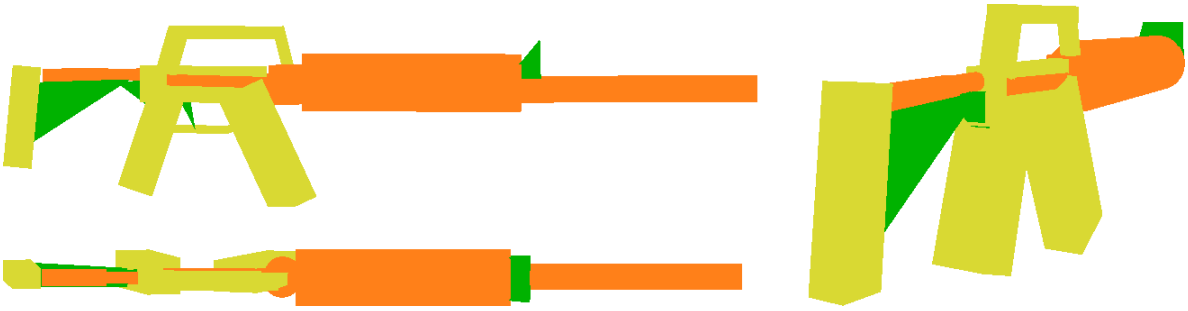


Figure 9.6: A rifle

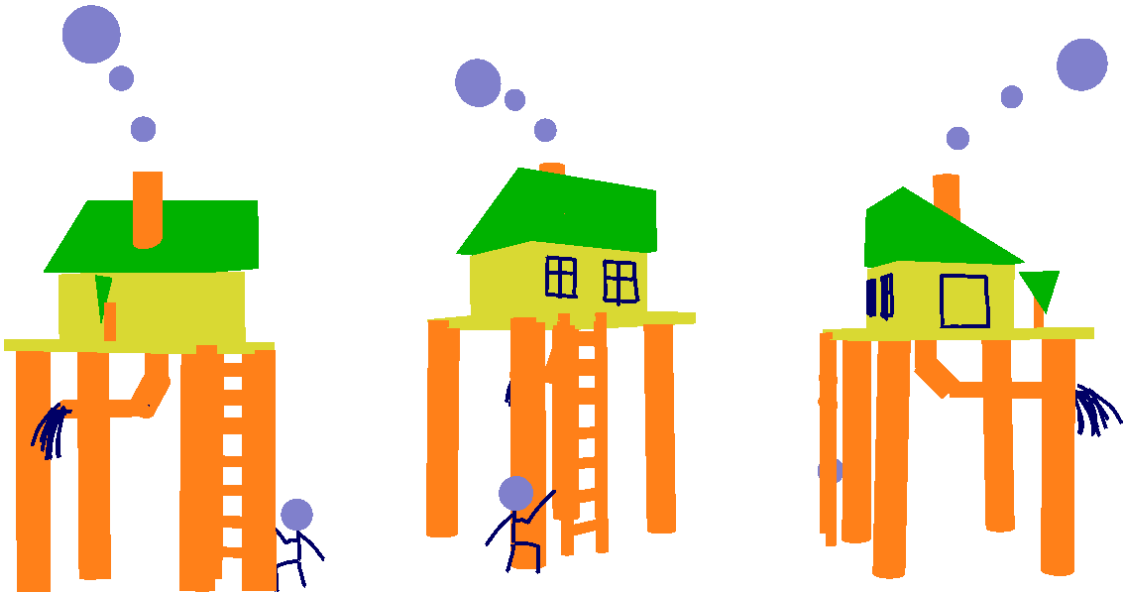


Figure 9.7: An elevated house