Towards a 3D Sketch-Based Modelling API

Yi Zeng **Zijiang Song** **Burkhard C. Wünsche**

Department of Computer Science
University of Auckland, Auckland, New Zealand,
Email: yzen015@aucklanduni.ac.nz, zson013@aucklanduni.ac.nz, burkhard@cs.auckland.ac.nz

Abstract

Sketch-based applications are rapidly gaining popularity in 3D modelling because of the intuitive pen-and-paper metaphor. Even inexperienced users with little computer graphics and digital design background can use them to create 3D models rapidly. However, the development of sketch-based applications is usually difficult and time consuming. In this paper, we present a framework for simplifying the development of sketch-based 3D modelling applications. The framework integrates existing techniques for 3D sketch processing with a processing pipeline for sketch input, a state-machine for defining processing parameters and modes, and a customised event handler. The modular design means that the functionality of the framework can be easily extended in the future. Experimental results suggest that the framework is easy to use and the implemented functionalities work correctly.

Keywords: sketch-based modelling, sketch API, sketch recognition, surface reconstruction

1 Introduction

Computer generated 3D models are common and important components of many virtual environments. They are used in a wide range of application fields including computer games, movies, medical simulations, robotics, architecture, urban design, and education. Professional modelling tools such as Maya and AutoCAD are powerful and able to construct realistic 3D model with high precision, but they have complex interfaces with a steep learning curve and are most suitable for expert users. In many applications low precision is acceptable and the emphasis is on having an intuitive modelling tool allowing untrained users to create 3D content quickly and easily.

Sketch-based modelling tools are a promising solution, since sketching is intuitive (pen-and-paper metaphor), gives complete freedom over the input, encourages creativity (Gross & Do 1996), facilitates problem solving (Wong 1992), and allows users to concentrate on the overall design of a 3D model rather than the modelling tool itself. With sketch interfaces even inexperienced users without graphics knowledge can create 3D content quickly (Yang & Wünsche 2010, Olsen et al. 2011).

Due to the rapid uptake of consumer-level touch screen devices, the number of sketch-based modelling applications has increased significantly over the past decade. However, developing a sketch-based 3D modelling application is still difficult and time consuming.

One reason for this is, that currently there is no general framework for 3D sketch processing. For each new application, developers have to spend a large amount of effort implementing their own version of the fundamental tasks in sketch processing, i.e. sketch smoothing, strokes combination, shape recognition, 3D projection etc.

In order to improve the efficiency and productivity of developing new sketch-based applications, we propose a 3D sketch-based modelling framework, which integrates common functionalities of existing 3D sketch-based modelling tools. The framework is fully extendible so that more features can be easily included.

Section 2 reviews previous work on sketch APIs and frameworks. In section 3 we review some examples of sketch-based 3D modelling applications. From these examples we identify common concepts and constraints, which are used in the requirement analysis presented in section 4. Section 5 presents the design of our framework and section 6 discusses implementation details. We evaluate the presented sketch API in section 7 and conclude the project and discuss potential future work in section 8.

2 Related Work

Despite of the popularity of 3D sketch-based modelling most existing APIs and frameworks only support 2D sketching. The arguably most widely known tools are the Microsoft Ink tools, which comprise the `Pen API` for capturing pen motion, the `Ink API` for rendering, grouping, storing and loading ink (pen motions), and the `Ink Analysis API` for handwriting recognition (Windows Dev Center 2013*a*,*b*).

Several tools have been presented, which facilitate the development of components of sketch-based modelling tools. For example, RATA simplifies the development of sketch recognisers (Plimmer et al. 2012).

A review of the literature resulted in the identification of only one API for 3D sketch-based modelling: The SketchUp Ruby API enables developers to extend the functionality of SketchUp as well as create macros to encapsulate complex tasks (Trimble Navigation Ltd. 2013). However, the API only allows interaction with traditional geometric entities, such as points, faces and meshes. The "raw" sketch input does not seems to be accessible to developers.

3 Background

We reviewed different sketch-based modelling applications and identified the following important components:

Copyright ©2014, Australian Computer Society, Inc. This paper appeared at the 15th Australasian User Interface Conference (AUIC 2014), Auckland, New Zealand, January 2014. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 150, Burkhard C. Wünsche and Stefan Marks, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

3.1 2D Sketch Processing

Rendered sketches look more attractive and analysis of 2D sketches and creation of 3D surfaces is made easier when using a simple mathematical representation for them.

Igarashi et al. evaluate each stroke input for potential geometric relations such as horizontal and vertical strokes, connections, alignments, and symmetry (Igarashi et al. 1997). Interactive beautification is performed after identifying the most suitable geometry relation.

Sezgin et al. eliminate noise from free-hand drawings by combining average based filtering and scale space filtering (Sezgin et al. 2001) . The method uses curvature information and pen speed data in order to differentiate between shape features of a curve and unintended wriggles.

A smooth curve can be obtained from sketch input by reducing the number of samples with the Douglas-Peucker algorithm and interpolating the resulting samples using a Catmull-Rom spline (Wünsche 2013).

3.2 Sketch Recognition

Creation of 3D geometry often requires knowledge of the type of shape a 2D sketch represents. Barber et al. use primitive shape properties such as length, oriented bounding box (OBB), curvature, direction changes, length-area ratio etc. in order to recognise geometric shapes such as straight lines, triangles, rectangles, and closed curves (Barber et al. 2010). Plimmer et al. use machine learning algorithms to develop sketch-recognisers (Plimmer et al. 2012).

3.3 3D Geometry Creation

3.3.1 Silhouette-Based Methods

The arguably most popular class of sketch-based 3D modelling techniques uses sketch input to represent the silhouette (outline) of a 3D object. The outline, usually referred to as contour, is expanded to a 3D object by making the assumption that the object is "blobby", i.e., the cross section of each component of the sketched contour is circular.

The 3D surface can be obtained by computing a skeleton of the contour and then fitting circular cross-sections around the skeleton (Igarashi et al. 1999, Igarashi & Hughes 2003, Levet & Granier 2007). More complex shapes can be obtained by sketching contours of local features (Zimmermann et al. 2008). Alternatively implicit surfaces can be used to convert contours to 3D bodies (Karpenko et al. 2002, Schmidt et al. 2006, de Araújo et al. 2004).

Two interesting application of silhouette-based algorithms are garment and tree modelling. For tree modelling the user sketches the outline of the crown of the tree and the algorithm computes a fitting branching structure based on existing templates and a probabilistic distribution (Chen et al. 2008). Garments can be modelled by sketching their outline and the algorithm automatically fits them to the body shape (Turquin et al. 2007).

3.3.2 Contour-Based Methods

Contours include all visible lines and divisions of a shape, e.g., discontinuities in the surface gradient. Contour information can be used to edit 3D meshes by making local modifications (Karpenko & Hughes 2006), or by using free form deformations to adapt the underlying 3D shape (Nealen et al. 2007). If domain specific information is know complete shapes can be obtained using only a few input sketches. For example, Gain et al. model complex 3D terrains by drawing the silhouette, spine and bounding curves of landforms (Gain et al. 2009).

A popular application of contour-based methods is the sketching of technical drawings and 3D CAD models. Computer designed items are often characterised by a blocky shape, flat or arced surfaces, sharp or evenly rounded edges and corners, many parallel and orthogonal edges and faces, and symmetrical features. These features can be captured using silhouettes which can then be interpreted using application specific constraints, e.g., surfaces of CAD objects frequently form 90 degree angles (Zeleznik et al. 1996, Eggli et al. 1997, Mitani et al. 2002).

3.3.3 Skeleton-Based Methods

Skeleton-based techniques are most popular for modelling complex fibrous and branching structures. Ijiri et al. (2005) use sketch input to model the stem and branches of flowers. The 3D shape of a stem is computed by solving a differential equation such that the curvature and appearance of the resulting 3D shape is identical to the 2D sketch.

A more general system is "Thor" (Arcila et al. 2008). The user draws a skeleton using a series of sketches. The initial sketch defines the main shape and subsequent sketches modify it. The user can draw a radius for each skeleton segment and a 3D surface is generated by fitting a generalised cylinder to it.

3.3.4 Cross Section-Based Methods

A less common approach for creating 3D models is to sketch 2D cross-sections. McCord et al. (2008) model orchids by allowing users to sketch the cross section of the labellum of an orchid, which is then expanded into a 3D surface by fitting ellipsoidal contours around.

Cross sections are used in most professional modelling tools, but are usually not sketched but represented by parametric curves. 3D surfaces are obtained by extrusion or by computing the tensor product with a second parametric shape. SketchUp employs some of these principles using an interface mixing sketch and CAD elements (Trimble Navigation Ltd. 2013).

Olsen et al. model 3D buildings from two-dimensional sketched cross-sections. The algorithm analyses the sketch input, extracts shape and detail information, predicts the building type, and creates 3D models by applying an extrusion, rotation of a projection algorithm (Olsen et al. 2011)

3.3.5 General Sketch-Based Modelling Tools

ILoveSketch is a 3D curve sketching system where users are able to draw curves freely in 3D space (Bae et al. 2008). Several tools are provided to select a drawing plane/surface, e.g., coordinate planes from user-defined coordinate systems and planes obtained by extruding a sketched curve. 3D curves can also be obtained from two 2D curves using epipolar geometry.

3.4 Sketched-Based Animation Systems

Sketch-based animation of objects can be achieved by sketching motion paths, which define the motion of the entire object (Steger 2004) or the motions of components of an object such as limbs (Schauwecker et al. 2011). Motion paths can also be subdivided into primitives and matched to pre-animated motions (Thorne et al. 2004).

An alternative solution is to sketch key frame poses, and use bone information (Davis et al. 2003) or body contour information (Mao et al. 2007) to infer 3D motions.

4 Requirements Analysis

Our analysis of successful sketch-based modelling systems for 3D objects shows that virtually all applications use the following steps:

- **Sketch simplification and beautification:** in order to analyse and interpret sketch input it has to be converted into a simplified mathematical form, e.g., a polyline or smooth curve.
- **Sketch recognition:** in order to create 3D geometry from 2D geometry, the geometric properties of input sketches must be identified, e.g., "sketch is a straight line", "sketch is a closed curve" or "sketch is a rectangle".
- **Context:** The interpretation of a 2D sketch often depends on the context. For example, a sketch drawn over a surface can indicate deformation of the surface, whereas a sketch drawn touching a surface can indicate an extrusion process.
- **Constraints:** Limiting the range of possible 3D shapes makes it easier to recognise and interpret 2D sketch input.

Our goal is to develop a 3D sketch API which simplifies the development of a large range of sketch-based modelling applications. We hence need the following functionalities:

- Functions for sketch simplification and beautification.
- Functions for sketch recognition.
- Functions to set context, e.g., modelling/edit/interaction mode, definition of sketch planes.
- Functions to select objects, e.g., closest existing sketch to a new sketch, closest 3D object to a sketch.
- Functions to modify objects, e.g., deformation of a surface using sketch input.
- Functions to derive 3D sketches from 2D sketch input, e.g., use depth values from a 3D object or sketch plane.

5 Design

In order to fulfill the identified requirements developers must be able to specify each stage of the sketch input processing and model generation. Developers must be able to store intermediate objects, such as surfaces defined from sketch input, and use them in subsequent interactions (e.g., deform a surface using sketch input).

Application developers must be able to define different functionalities depending on user input. We hence need to provide event handling. The event handler evaluates each sketch and generates corresponding events, e.g., if the drawing of a sketch was completed, a closed sketch was detected, or a sketch was drawn over an existing object.

In order to minimise the developer's workload and reduce code redundancies we provide functions to set states, similar to the OpenGL graphics API. For example, the developer can select a function for sketch simplification and all subsequent sketches are processed accordingly, until that function is changed.

Our Sketch API consists hence of a sketch processing pipeline, a state machine, event handler, and object database as illustrated in figure 1.

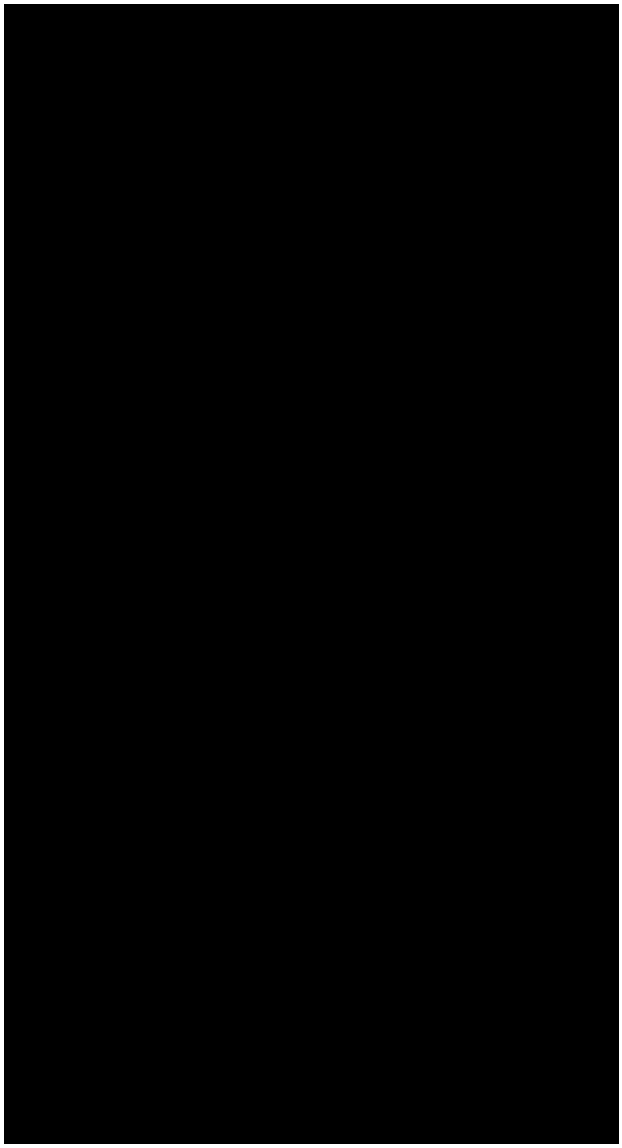

Figure 1: The Sketch API consists of a sketch processing pipeline, and a state machine, event handler, and object database.

5.1 Sketch Processing Pipeline

The core of our framework is the sketch processing pipeline shown on the right-hand side of figure 1. Whenever a sketch is detected, it is processed through the pipeline. Each stage is configurable via setting an appropriate mode and can be enabled or disabled. For example, if a developer wants to simplify sketch input using the Douglas-Peucker algorithm and then project the 2D sketch onto the predefined 3D drawing plane, then this can be achieved using only three lines of code:

```
setSimplificationMode(DOUGLAS_PEUCKER);
setProjectionMode(PLANE);
setPipelineMode(PROJECTION |
                SIMPLIFICATION);
```

5.1.1 Sketch Input

In order to make the API as flexible as possible it should work for different input devices such as mouse, touch screen, and drawing tablets. We hence use an extra layer of abstraction, which converts device specific input into a format suitable for our API. For example, when using the mouse as input device mouse

events such as "mouse up" and "mouse down" are converted into "sketchStart" and "sketchEnd" events and the mouse coordinates between any such event sequence are converted into a sequence of 2D points with duplicates removed. When using a different input device only this abstraction layer needs to be modified.

At this point we do not yet record additional parameters such as "drawing speed" and "pen pressure" (which could be simulated with the mouse wheel).

5.1.2 Sketch Simplification

Sketches often contain unintended jags and other errors. In order to simplify the sketch input we allow developers to use the Douglas-Peucker algorithm (Douglas & Peucker 1973). Sketches are smoothed using Catmull-Rom spline interpolation (Catmull & Rom 1974).

5.1.3 3D Projection

Finding the correct 3D positions for a 2D input sketch is arguably the most important step in 3D sketch-based modelling. In our framework, we find suitable z-coordinates for a 2D sketch by projecting the sketch onto user-defined planes and surfaces. Currently we support the following 3D mapping modes:

- **3D Plane:** The sketch is projected on a specified 3D drawing plane. The drawing plane can be defined by the developer or can be selected by the user through the built in "Plane Selector" widget.
- **Extruded Surface:** The sketch is projected on a 3D extruded surface, which is obtained by extruding a sketch along a vector. The surface can be specified by the developer, e.g., a curved sketch followed by a straight line connected to it. Alternatively an extruded surface can be specified at run time by the user by using the built-in "Surface Selector".
- **Arbitrary Surface:** The sketch is projected onto an arbitrary 3D surface by using the surface's z-coordinates.

More details are given in section 6.

5.1.4 Sketch Assembly

The sketch assembly step provides functions to group sketches, e.g., combine multiple strokes into a single sketch, close a sketch, or form meaningful shapes such as arrows.

5.1.5 3D Sketch Recognition

In this stage the assembled sketch is classified into a set of predefined shapes, such as rectangle, triangle, circle, straight line, scribble etc. The developer can enable automatic beautification of shapes, e.g., replacing a sketched circle with a perfect parametric circle. Our current implementation contains a simple approximation for this, but better algorithms for finding the optimal fitting geometric shape have been described in the literature (Arvo & Novins 2000).

5.1.6 3D Object Generation

The last step in the processing pipeline is the generation of 3D surfaces and objects from the input sketch. Currently, the framework supports the following modes:

- **Extrusion:** Extrude a 3D stroke in a specified direction.
- **Filling:** Create a parametric surface from a closed sketch.

5.2 Event Handling

Developers can specify complex functionalities by using sketch events. The current prototype supports the following events:

- **Sketch Begin:** Triggered when a sketch begins.
- **Sketching:** Triggered repeatedly as long as a sketch is still drawn.
- **Sketch End:** Triggered when a sketch ends.
- **Sketch Closed:** Triggered when a closed sketch is detected.
- **Drawing Plane Changed:** Triggered when the drawing plane is changed.

In order to prevent conflicts with the event handler of the underlying graphics library (GLUT) we create our own mouse callback functions which the developer can call using `sketchMouseFunc(handler)`.

6 Implementation

In this section we explain some of the key functionalities in more detail.

Figure 2: Projection of a 2D sketch onto a plane in 3D.

6.1 3D Projection

Sketch raw data consists of 2D screen coordinates. The 3D coordinates of a 2D sketch are determined as follows:

If the current OpenGL state associates the sketch with a sketch plane in 3D, we cast rays from the view point through the sketche's screen coordinates and compute the intersection points with the sketch plane. Figure 2 shows an example.

An extruded surface is constructed by extruding a 3D sketch along a 3D vector. If the current OpenGL state associates the sketch with such an extruded surface we can compute its 3D coordinates similar as above. This is possible since a 3D sketch is approximated by a sequence of line segments and the resulting extruded surface is hence a quadstrip (sequence of rectangles). Figure 3 shows an example.

In many instances we want to sketch on an arbitrary 3D object. In order to make the algorithm as general as possible our only requirement is that the 3D object can be rendered with depth-buffer values. Examples are polygon meshes, dense point clouds, or

Figure 3: Projection of a 2D sketch onto an extruded surface.

ray traced implicit surfaces. If the current OpenGL state associates the 2D sketch with such an object, we can compute its 3D representation by rendering the 3D object and retrieving for each 2D coordinate of the sketch the corresponding depth buffer value. Figure 4 shows an example.

Figure 4: Projection of a 2D sketch onto a renderable 3D object.

6.2 Shape Recognition

Our simple shape recogniser can currently detect rectangles, circles and scribbles (for deleting a sketch).

Rectangles are identified by first detecting whether the sketch is a closed shape (i.e., end points are close together relative to the bounding box size), simplifying the sketch with the Douglas-Peucker algorithm using a high error value obtained from the bounding box size, and then computing the number of turning points.

Circles are identified by first detecting whether the sketch is a closed shape, computing the centre of the bounding box, and checking whether all sketch points have a roughly equal distance to the centre point.

A scribble is characterised by an approximately equal number of significant turning points (angle $\geq$ 120°). Whenever a scribble is detected, the sketches covered by the bounding box of the scribble are immediately deleted.

Figure 5 shows an example.

6.3 NURBS Surface from Closed Sketch

NURBS surfaces are common in 3D modelling applications since they can be easily controlled using a control point mesh and knot vector, they have a high-level of continuity (smoothness), discontinuities can be inserted if desired, they have local control, and they are supported by most graphics APIs such as OpenGL.

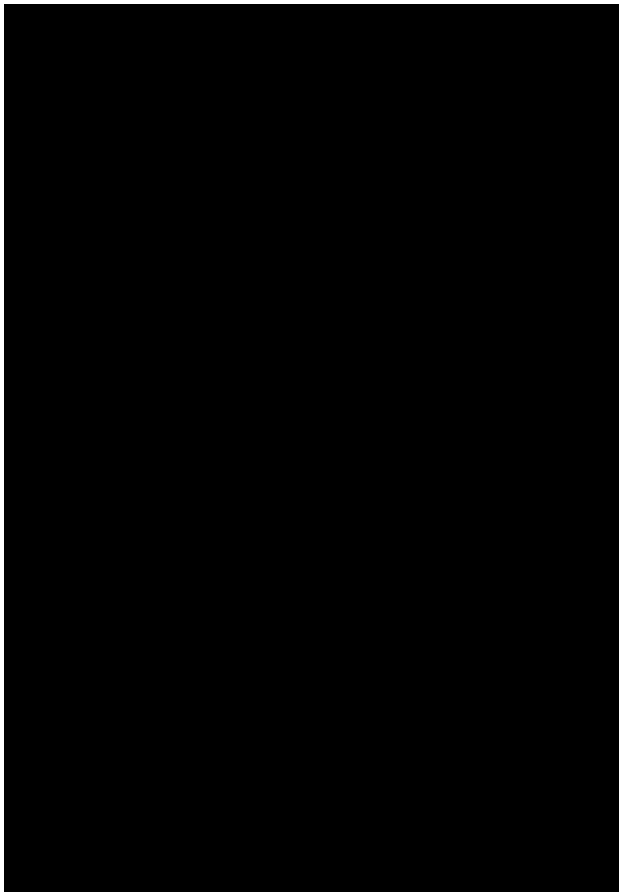

Figure 5: Recognition and beautification of a rectangle (top) and circle (bottom) and a scribble for deleting sketched objects (bottom).

We create a NURBS surface from a closed sketch by computing its oriented bounding box, using it to define the control point mesh, and then trimming the resulting surface using the sketch such that only the surface section inside the sketch remains (see figure 6). The resulting surface can then be modified using sketch input by adjusting control points accordingly.

Figure 6: A NURBS surface defined by a closed sketch.

7 Results

In this section we evaluate the effectiveness and robustness of the presented sketch API.

7.1 Effectiveness

The primary goal of our Sketch API is to enable developers to easily create 3D sketch-based modelling tools. In order to evaluate our tool we implemented

a simple tool for sketching 3D leaves. The modelled functionality has been previously been presented in different flower modelling tools (Ijiri et al. 2005, McCord et al. 2008). A leaf is sketched in three steps as illustrated in figure 7.

Figure 7: A leaf is sketched in three steps: (1) Sketch the outline of the leaf using one or two strokes and fit a NURBS surface to the closed sketch (left); (2) sketch a modifier stroke and project it onto the NURBS surface (middle); (3) compute the distance between the modifier stroke and the leave's centre line and warp the NURBS surface in the direction of the surface normal accordingly.

Using our framework the implementation of this functionality is straight forward.

The sketch processing pipeline requires only the stage 3D Projection and Sketch Assembly. 3D Projection is used to project the sketch onto the drawing plane, and Sketch Assembly is needed to connect multiple strokes forming the leaf. We also need to set up callback functions for sketch events. The resulting code is:

```
// sketch pipeline
setPipelineMode(SKETCH_PROJECTION |
                SKETCH_ASSEMBLY);
setProjectionMode(PROJECTION_PLANE);
setAssemblyMode(ASSEMBLY_ENDPOINS);
// callbacks
onSketchEnd(handleSketchEnd);
onClosedShapeDetected(handleClosedShape);
```

These handlers are called when corresponding sketch events are triggered. In our example a NURBS surface is created when a closed shape (leaf's outline) is drawn, and the surface is deformed in 3D space when a modifier stroke is drawn. The resulting code is:

```
void handleClosedShape(Stroke& stroke) {
  // when a closed sketch is detected
  myNURBS = new NURBSSurface(stroke);
  bClosedShapeDrawn = true;
  setPipelineMode(SKETCH_PROJECTION);
}

void handleSketchEnd(Stroke& stroke) {
  // when a stroke is completed
  if (bClosedShapeDrawn)
    myNURBS->deformSurface(stroke);
}
```

The last step is to draw the resulting NURBS surface:

```
if (myNURBS)
  myNURBS->drawSurface();
sketchDisplay();
```

Our results so far indicate that the framework is easy to use and suitable for a wide range of applications. Functionalities are currently very limited, but new ones are added each time we use the tool for novel application. For example, we currently work on using sketch input to complete 3D models obtained from point cloud data.

7.2 Robustness

In the current version of the framework, most of the implemented functionalities are working correctly, such as sketch simplification, projecting sketches onto a drawing plane, creating NURBS surface etc. However, the implementations of some functionalities are not robust, and the processing pipeline can fail as a result. We have identified two issues below:

Projection on Arbitrary Surface

This functionality is achieved by utilising the OpenGL depth buffer. In orthographic projection mode, the algorithm works correctly, because the depth value is a linear function, which means the depth value is accurate in all depth ranges. However, when using a perspective projection problems can occur if the near plane is set too close to the camera. In this case the depth buffer values form a non-linear function and pixels representing objects close to the near plane have a z-value with high precision, and pixels representing objects close to the far plane have a z-value with low precision. In the latter case the 3D coordinates of a sketch drawn over such an object are very inaccurate.

Shape Recognition

The shape recogniser uses a variety of threshold values, e.g., to determine whether a sketch is closed or whether it is a rectangle. We tried to make threshold values work correctly for a large variety of shapes by taking into account the size of a shape. However, many of these decisions are subjective and application dependent, i.e., what is a closed curve to one user might be an open curve to another one. Allowing the developer or user to set these parameters is not desirable, since it would significantly increase the complexity of the tool. A possible solution is to use a machine learning algorithm similar to (Plimmer et al. 2012).

8 Conclusion and Future Work

Sketch-based modelling is an exciting technology with a wide range of applications. By reviewing the current state-of-the-art and evaluating a variety of existing sketch-based modelling applications, we have designed and implemented a framework for 3D sketch-based modelling which integrates basic functionalities of 3D sketch processing.

We have tested the framework by using it to write a simple sketch-based modelling applications. Preliminary results suggest that the framework is easy to use, the implemented functionalities work correctly, and that it can be easily extended.

Future work will concentrate on adding more functionalities, improving the processing pipeline, and performing more extensive usability testing with more complex application scenarios and participants unfamiliar with the tool.

References

Arcila, R., Levet, F. & Schlick, C. (2008), Thor: Sketch-based 3d modeling by skeletons, *in* 'Smart Graphics', pp. 232–238.

Arvo, J. & Novins, K. (2000), Fluid sketches: continuous recognition and morphing of simple hand-drawn shapes, *in* 'Proceedings of the 13th annual ACM symposium on User interface software and technology (UIST '00)', ACM, pp. 73–80.

Bae, S.-H., Balakrishnan, R. & Singh, K. (2008), Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models, *in* 'Proceedings of the 21st annual ACM symposium on User interface software and technology (UIST '08)', ACM, pp. 151–160.

Barber, C. M., Shucksmith, R. J., MacDonald, B. & Wünsche, B. C. (2010), Sketch-based robot programming, *in* 'Proceedings of Image and Vision Computing New Zealand (IVCNZ 2010)', pp. 1–8.

Catmull, E. & Rom, R. (1974), 'A class of local interpolating splines', *Computer Aided Geometric Design* pp. 317–326.

Chen, X., Neubert, B., Xu, Y.-Q., Deussen, O. & Kang, S. B. (2008), Sketch-based tree modeling using markov random field, *in* 'ACM SIGGRAPH Asia 2008 papers', ACM, pp. 1–9.

Davis, J., Agrawala, M., Chuang, E., Popović, Z. & Salesin, D. (2003), A sketching interface for articulated figure animation, *in* 'SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation', Eurographics Association, pp. 320–328.

de Araújo, B., Jorge, J., Sousa, M. C., Samavati, F. & Wyvill, B. (2004), MIBlob: a tool for medical visualization and modelling using sketches, *in* 'SIGGRAPH '04: Posters', ACM Press, p. 107.

Douglas, D. & Peucker, T. (1973), 'Algorithms for the reduction of the number of points required to represent a digitized line or its caricature', *The Canadian Cartographer* **10**(2), 112–122.

Eggli, L., Ching-Yao, H., Bruderlin, B. D. & Elber, G. (1997), 'Inferring 3d models from freehand sketches and constraints', *Computer-Aided Design* **29**(2), 101–112.

Gain, J., Marais, P. & Strasser, W. (2009), Terrain sketching, *in* 'I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games', ACM, pp. 31–38.

Gross, M. D. & Do, E. Y.-L. (1996), Ambiguous intentions: a paper-like interface for creative design, *in* 'Proceedings of the 9th annual ACM symposium on User interface software and technology (UIST '96)', ACM, New York, NY, USA, pp. 183–192.

Igarashi, T. & Hughes, J. F. (2003), Smooth meshes for sketch-based freeform modeling, *in* 'I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics', ACM, pp. 139–142.

Igarashi, T., Matsuoka, S., Kawachiya, S. & Tanaka, H. (1997), Interactive beautification: a technique for rapid geometric design, *in* 'Proceedings of the Symposium on User Interface Software and Technology', pp. 105–114.

Igarashi, T., Matsuoka, S. & Tanaka, H. (1999), Teddy: a sketching interface for 3d freeform design, *in* 'Proceedings of SIGGRAPH '99', ACM Press, pp. 409–416.

Ijiri, T., Owada, S., Okabe, M. & Igarashi, T. (2005), 'Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints', *ACM Trans. Graph.* **24**(3), 720–726.

Karpenko, O. A. & Hughes, J. F. (2006), 'Smoothsketch: 3d free-form shapes from complex sketches', *ACM Transactions on Graphics* **25**(3), 589–598.

Karpenko, O., Hughes, J. & Raskar, R. (2002), 'Free-form sketching with variational implicit surfaces', *Computer Graphics Forum* **21**(3), 585–594.

Levet, F. & Granier, X. (2007), Improved skeleton extraction and surface generation for sketch-based modeling, *in* 'GI '07: Proceedings of Graphics Interface 2007', ACM, pp. 27–33.

Mao, C., Qin, S. F. & Wright, D. (2007), Sketch-based virtual human modelling and animation, *in* 'SG '07: Proceedings of the 8th international symposium on Smart Graphics', Springer-Verlag, pp. 220–223.

McCord, G., Wünsche, B. C., Plimmer, B., Gilbert, G. & Hirsch, C. (2008), A pen and paper metaphor for orchid modeling, *in* 'Proceedings of the 3rd International Conference on Computer Graphics Theory and Applications (GRAPP 2008)', pp. 119–124.

Mitani, J., Suzuki, H. & Kimura, F. (2002), '3d sketch: sketch-based model reconstruction and rendering', pp. 85–98.

Nealen, A., Igarashi, T., Sorkine, O. & Alexa, M. (2007), Fibermesh: designing freeform surfaces with 3d curves, *in* 'SIGGRAPH '07: ACM SIGGRAPH 2007 papers', ACM, p. 41.

Olsen, D. J., Pitman, N. D., Basak, S. & Wünsche, B. C. (2011), Sketch-based building modelling, *in* 'Proceedings of GRAPP 2011', pp. 119–124.

Plimmer, B., Blagojevic, R., Chang, S. H.-H., Schmieder, P. & Zhen, J. S. (2012), RATA: codeless generation of gesture recognizers, *in* 'Proceedings of the 26th Annual BCS Interaction Specialist Group Conference on People and Computers (BCS-HCI '12)', British Computer Society, pp. 137–146.

Schauwecker, K., van den Hurk, S., Yuen, W. & Wünsche, B. (2011), Sketched interaction metaphors for character animation, *in* 'Proceedings of GRAPP 2011', pp. 247–252.

Schmidt, R., Wyvill, B., Sousa, M. C. & Jorge, J. A. (2006), Shapeshop: sketch-based solid modeling with blobtrees, *in* 'SIGGRAPH '06: ACM SIGGRAPH 2006 Courses', ACM, p. 14.

Sezgin, T. M., Stahovich, T. & Davis, R. (2001), Sketch based interfaces: Early processing for sketch understanding, *in* 'Proceedings of the Workshop for Perceptive User Interfaces (PUI 01)', ACM Press, pp. 1–8.

Steger, E. (2004), Sketch-based animation language, Technical report, Department of Computer Science, University of Toronto. URL: `http://www.cs.toronto.edu/~esteger/sketchlang/index.html`.

Thorne, M., Burke, D. & van de Panne, M. (2004), 'Motion doodles: an interface for sketching character motion', *ACM Trans. Graph.* **23**(3), 424–431.

Trimble Navigation Ltd. (2013), 'Sketchup ruby api - creating geometry'. http://www.sketchup.com/intl/en/developer/docs/gsrubyapi_examples, Last retrieved 24th August 2013.

Turquin, E., Wither, J., Boissieux, L., Cani, M.-P. & Hughes, J. F. (2007), 'A sketch-based interface for clothing virtual characters', *IEEE Comput. Graph. Appl.* **27**(1), 72–81.

Windows Dev Center (2013*a*), 'Ink analysis overview'. http://msdn.microsoft.com/en-us/library/windows/desktop/ms704040\%28v=vs.85\%29.aspx, Last retrieved 24th August 2013.

Windows Dev Center (2013*b*), 'Microsoft.ink'. http://msdn.microsoft.com/en-us/library/ms826516.aspx, Last retrieved 24th August 2013.

Wong, Y. Y. (1992), Rough and ready prototypes: lessons from graphic design, *in* 'Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems (CHI '92)', ACM, New York, NY, USA, pp. 83–84.

Wünsche, B. C. (2013), 'Compsci 373 assignment 2 sammple solution - question 3: Sketch-based modelling'. http://www.cs.auckland.ac.nz/courses/compsci373s1c/assignments/CS373Assignment2_SampleSolution.pdf, Last retrieved 24th August 2013.

Yang, R. & Wünsche, B. C. (2010), Life-sketch: a framework for sketch-based modelling and animation of 3d objects, *in* 'Proceedings of the Eleventh Australasian Conference on User Interface (AUIC '10)', Australian Computer Society, Inc., pp. 61–70.

Zeleznik, R. C., Herndon, K. P. & Hughes, J. F. (1996), SKETCH: An interface for sketching 3d scenes, *in* 'Proceedings of SIGGRAPH '96', ACM Press, pp. 163–170.

Zimmermann, J., Nealen, A. & Alexa, M. (2008), 'Sketching contours', *Computers & Graphics* **32**(5), 486–499.