Freeform spatial modelling using depth-sensing camera

Mario Volarević, Petar Mrazović, Željka Mihajlović
Faculty of Electrical Engineering and Computing, Zagreb, Croatia
mario.volarevic@gmail.com, petar.mrazovic@gmail.com, zeljka.mihajlovic@fer.hr

Abstract - This paper proposes a novel 3D direct interaction modelling system for manipulating voxel based objects using currently available 3D motion sensing input devices such as Microsoft Kinect. The guiding principle while developing the system was to imitate natural human modelling behaviour and provide a real life experience of 3D object manipulation, inspired by the techniques used in modelling clay. Properties of a functional prototype application are presented and software architecture of the created tool is analysed. Descriptions of newly developed algorithms are included, grouped into several categories. Visualization algorithms are used for defining properties and creating usable modelling mass from volumetric models. On the other hand, algorithms related to object recognition and human computer interaction include various techniques for depth segmentation, contour detection, finger recognition and virtual control with gestures.

I. Introduction

Computers of various sizes and forms surround us in everyday life and it is hard to imagine life without them, but in some cases interaction is still somewhat unintuitive because methods of indirect control are used. Usually, we first have to learn how to use the controller and only then can we perform the desired task. Notable examples include using a mouse to control a pointer and activate onscreen actions or using gamepads or joysticks to control the environment with different buttons and sticks. Rapid development of new and interesting technologies related to human-computer interactions encouraged us to research and develop a novel interaction model which could improve our experience when interacting with 3D objects in a computer generated world. The key motivation behind technologies which use direct manipulation is not only to naturally represent human actions, but also to promote the notion of people performing tasks themselves, and not through some intermediary, like a computer. Problem with unnatural controls is especially apparent when doing 3D modelling since usually 2D controllers are used for manipulating objects in 3D world. One of the possible solutions for performing these tasks in a more natural way will be presented in this paper. Final goal is to have a functional prototype which would be more like a real life experience, for instance, as you would do clay modelling. Therefore, suitable hardware for direct 3D control has to be found and accompanying software support has to be developed.

In the last couple of years situation with direct control in consumer devices is getting better because smartphones and tablets became widely available and the main method of interaction is via touchscreen surfaces. Touch interfaces allowed users to use movements from real life (like translations) to be directly mapped to corresponding actions for onscreen elements. There are some programs like Autodesk 123D Sculpt on iOS which use this kind of interaction but unfortunately, with current technology, there is no depth information so operations like extruding or making holes in the model feel unnatural and don't provide good feedback. Reconstruction of depth information can be somewhat alleviated by adding gyroscopes and accelerometers which provide limited 3D movement recognition. One of the best examples of such controller is Nintendo Wiimote because of its high precision, wide availability, user friendliness and popularity. Drawback is that the controller has to be held in a user's hand which means that fingers can't be used for modelling with a fine level of detail [1, 2]. Also, our goal is to provide a truly natural user interface without any additional devices that would be a burden to a user.

An alternative approach is offered by Microsoft with Kinect where a user's own body acts as a controller. The advantage in this case is the ability to use the full 3D range of movement "without" constraints[1] and have free hands to interact with the virtual world in a natural way, and that was one of the initial prerequisites. Another device with similar characteristics, which could be also suited for the 3D modelling methods described here, is Leap Motion Controller. Since the mode of operation and detection is similar compared to Kinect, it should be relatively easy to adapt the developed prototype for use with Leap.

This paper is divided into six sections. Section 2 investigates related work where relationship with our prototype is based on similar ideas or a similar controller was used. Section 3 consists of several subsections where technical implementation details of used algorithms are described. In section 4, both performance and functional results are presented and discussed. Section 5 provides insight into possible future development of the prototype and finally, section 6 concludes the paper.

II. Related work

During the development of our prototype, we focused our research on a combination of computer modelling and

[1] There are still some constraints like usable sensor range or speed with which the user moves the hands.

motion gesture interfaces. Our proposed 3D direct interaction modelling system was significantly inspired by the work of Cho et al. [3] presented in the paper under the title “Turn: A Virtual Pottery by Real Spinning Wheel” and later expanded journal article [4]. Cho proposes a digital pottery system that successfully reflects real-world gestures to their virtual counterparts. The study focuses on the specific aspect of clay modelling where the freedom of movement and possible creations are limited by a spinning-wheel modelling technique. On the other hand, the interaction model proposed in our paper tries to implement virtual freeform modelling and object manipulation without spatial restrictions or any predefined movements.

In the paper “Gesture Interface for 3D CAD Modelling using Kinect”, Dave et al. [5] successfully bring the intuitiveness of motion gestures to a CAD environment using Kinect in combination with constructive solid geometry and deformations, which produces results similar to traditional clay modelling. However, the restriction of the proposed creative solution is that Kinect could be only used to control tools which are already present in standard CAD programs. Using their prototype as an inspiration, we tried to remove any kind of restriction that could be a potential obstacle in providing a real life experience of freeform modelling in its true sense. Oliver Kreylos [1,2] also created modelling prototypes with motion input devices, but these prototypes were using motion controllers like Razer Hydra and Wiimote. Oliver’s presented work did not try to achieve direct interaction via natural user interfaces, but instead, it utilized benefits of motion controllers to develop highly accurate and usable modelling system. The evaluated results are very impressive, and therefore present great challenges to be overcome, compared to our modelling system with less accurate, object recognition based, methods, i.e. without motion controllers.

Finger recognition techniques used in this paper were partially based on the algorithms described by Daniel James Ryan [6] in his thesis. Although it was possible to directly use the algorithms as they were described in his thesis, we decided to adapt and optimize recognition algorithms to better accommodate our requirements where main reasons for modifications were strict performance requirements.

III. Proposed solution

Our solution can be divided into several distinct components which, when combined, achieve the final goal of having a real-time system which allows completely freeform 3D modelling that also imitates real life clay modelling behaviour. Basic components of the proposed solution are representation of the virtual clay, the finger recognition system and the visualization and interaction framework.

A. Virtual clay

The first component of the proposed solution defines the properties of the virtual modelling clay compound. An approach based on voxels was chosen because while analysing different alternatives for mesh representation this seemed like the most natural solution when dealing with multiple simultaneous extrusions and intrusions. The marching cubes algorithm [7] was used to render the volumetric model, but in a GPU accelerated variant, to make the prototype sufficiently fast to be executed in real-time. GPU acceleration could have been achieved using some GPGPU APIs, like OpenCL or CUDA, but since our goal was also to provide support for older computers, a more compatible approach using shaders was chosen. The method used for GPU acceleration of marching cubes was based on the guidelines from the NVidia’s GPU Gems article [8]. The main idea is to load the model in a 3D texture and then use GPU shaders to process all cubes simultaneously. A version with multiple shader passes was used, which enabled moving as much computations as possible from the geometry shader to the vertex shader. Reducing the workload in the geometry shader was required because that shader is extremely slow compared to other shaders.

Two passes were used, where coordinates of the corners of the voxel were the initial input and rendered polygons were the final output. The first pass in the vertex shader calculates positions of the voxels and their polygon configurations. Voxel positions and polygon configurations are then sent to the geometry shader where empty voxel configurations are rejected and simple markers for relative positions of triangle vertices are generated. Up to five markers are generated since five is the maximum number of polygons per voxel. Markers are then used in the vertex shader of the second pass where real 3D world space polygons coordinates are calculated, as well as normals. All of these previous steps led to an optimization which improved performance because it reduced the workload on geometry shader. The shader just had to stream out newly generated vertices and did not have to calculate anything. Finally, pixel shader is used to calculate lightning and render the model with proper colours.

B. Finger recognition

The second component in the developed system is related to finger recognition and it is based on Microsoft Kinect combined with algorithms for detecting and tracking body and various joints. The official Kinect API provides support for obtaining 3D coordinates of major joints so hand positions could be easily found, but for coordinates of individual fingers custom segmentation and recognition algorithms had to be developed and this process can be divided into several stages. In the first stage Kinect API is used to roughly detect the centre of the hand, and then, we propose algorithm based on the coordinates of the detected centre hand points. The image around the detected points is segmented by depth and hand contours are extracted. Segmentation was based on the assumption that hands are always in front of the body so it could be performed more easily.

Since the obtained contour is saved in a 2D field, it has to be converted to a 1D aligned array so it could be used with a k-curvature algorithm that is used to detect fingertips. Conversion was done with a custom algorithm which collects neighbouring contour points using circular search pattern where starting search position for the next

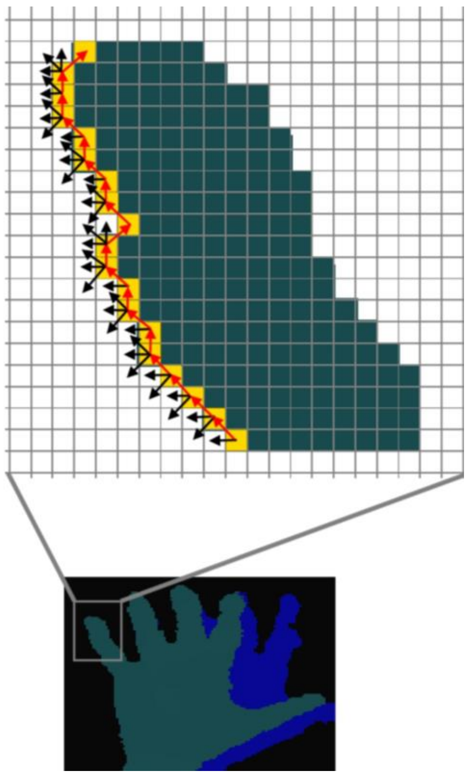

Figure 1. Execution of the contour detecting algorithm. Arrows show how the algorithm advanced while searching for the contour. Black arrows point to rejected background pixels and red arrows show the direction of contour pixels.

point depends on the positions of the current and previously found points. Figure 1 shows a simplified example of the contour-finding algorithm. In the extracted contour, k-curvature calculates the fingertip positions by taking three points from the array separated by factor k. Curvature and centroid calculations performed on the selected three points are used as features for detection of fingertips, as seen on Figure 2. The proposed algorithm works with the assumption that hands are mostly curved smoothly compared to the fingers, and centroid calculation is used to distinguish fingertips from valleys between fingers because these are the only places with sharp angles.

After finger positions are calculated, they have to be converted to Kinect's projection space coordinates so they are comparable with real world dimensions. Coordinate conversion was done using methods from Kinect API. After the final feature positions have been obtained, they were transferred to the interaction system where they were processed and converted into gestures.

C. *Interaction system*

Finally, there is a visualization step and interaction system that connects the other two, previously described, components and provides an input/output interface for the user. It is implemented using SharpDX Toolkit API, which is based on DirectX 11 and can be considered as an unofficial successor to Microsoft XNA Framework. SharpDX is currently the fastest managed DirectX APIs and is just 1.5x times slower than native code. It also forms a basis for many graphics engines, of which MonoGame is the most famous one.

Primary output is visual and is represented by a window where volumetric model with its bounding box is displayed. In the main window, there is also an abstract model which uses simple spheres to represent hands and fingers. Also, to ease the positioning in front of Kinect, because of the limited range and viewport, an additional sub-window is displayed, which shows depth and finger recognition data.

Hand gestures are used as primary input and were designed to be as intuitive as possible, but it still takes some time to get used to controls since there is no force feedback. Gestures are defined in the following way: if hands are not touching the model, it is possible to adjust the view by closing them (like grabbing an object) and then move, rotate and zoom it, and if they are inside the model or touching it, virtual clay is either added or removed on coordinates calculated based on the position of the fingertips. Some possible manipulations while using the program can be seen in the images in Figure 3. Additionally, keyboard controls can also be used for view manipulation, as a backup for cases when Kinect isn't connected or stops functioning properly due to bad lighting.

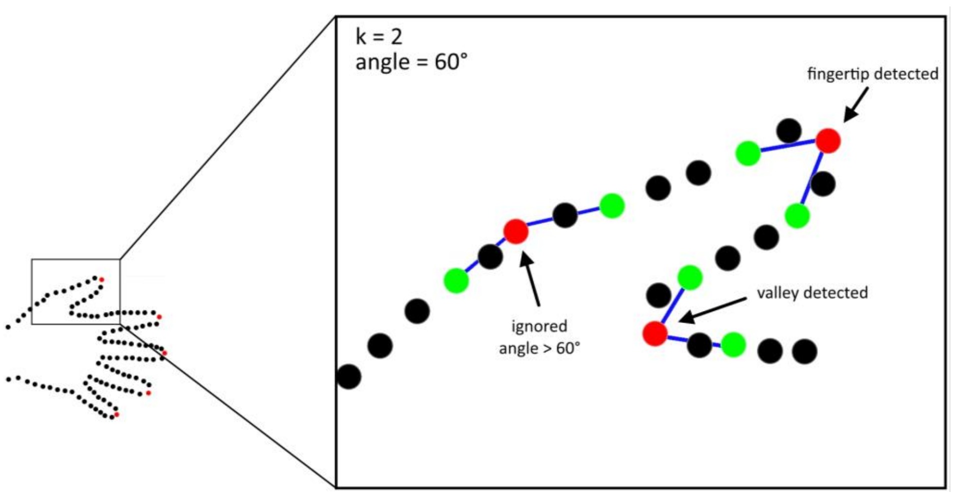

Figure 2. Simplified representation of the k-curvature algorithm. Fingertips are detected based on the angle between three points separated by factor k. Valleys as false positives are ignored because their centroid isn't located inside a finger

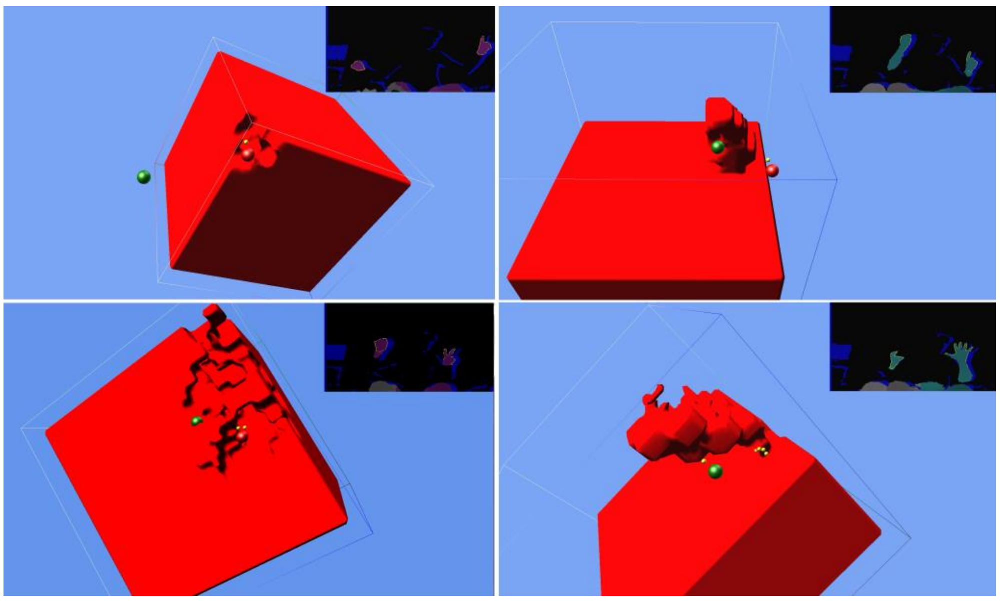

Figure 3. Several images of a working program where various intrusions, extrusions and view manipulations on a model can be observed

IV. RESULTS

The developed prototype application showed that results are promising, both considering performance and functionality. Even on relatively underpowered hardware, represented by Configuration 1, compared to other listed configurations, it is possible to track the fingers and render the model with reasonable quality in real-time with all, initially planned, features enabled. There were several hardware configurations on which this program was tested to show its scalability and also good compatibility, considering the variety of the used components. Table 1 lists relevant parts of the specifications for each of the configurations. Since the host part of the program is single-threaded and the main computation, which is multi-threaded, is executed on the graphics card, turbo frequencies will be listed for the CPUs, and the number of cores with their frequencies for the GPUs, so scaling can be more easily evaluated.

Performance results were measured on a cubic model with Kinect connected and finger recognition turned on. An example of the model and the pose used for evaluation are shown in Figure 4. Samples were taken for several resolutions to show which cases are usable in practice for different configurations and how the algorithm scales depending on the complexity. A numerical representation of the results and used resolutions can be seen in Table 2. In Figure 5, which shows graphical representation of the results, it can be seen that drawing speed, measured in frames per second, scales as expected depending on the GPU architecture and considering the small differences between CPU performances it can be concluded that this prototype is GPU bound.

While measuring performance results where Kinect was active were not much different than results with Kinect inactive. There was a slight performance hit, but it was observable only on smaller resolutions, which means that image processing for Kinect took at most 5-10 ms of GPU time. Memory usage was also observed in both cases and results showed that when Kinect is not connected, each time when model resolution is changed, it steadily increases from 50 MB to 320 MB, but when garbage collection activates, after all assets have been loaded, transferred to GPU and not needed on CPU side anymore, it drops to around 60 MB. With Kinect connected, memory usage is constant and floats around 200 MB, even after garbage collection has activated.

Functionality-wise, gestures and model manipulations are working well while Kinect detects user joints correctly, but quite often it does not. One of the cases when Kinect does not detect joints at all is when users leave the Kinect detection range. The detection range is limited to distances between 0.8–3.5 m. This usually happens when moving hands closer to Kinect to change farther parts of the model and it could be avoided if users would not stand too close to Kinect, but in that case another problem arises because Kinect's resolution is not high enough and the algorithm which detects fingers does not work well anymore. In our case, experiments showed

TABLE I. HARDWARE CONFIGURATIONS USED FOR TESTING

	Configuration 1	Configuration 2	Configuration 3
CPU	Core2Duo T7500, 2.4 GHz	Core i5-430M, 2.53 GHz	Core i7-3630QM, 3.4 GHz
GPU	Radeon HD 3650m, 120x600 MHz	Radeon HD 5650m, 400x550 MHz	Quadro K2000M, 384x750 MHz
RAM	4 GB DDR2, 800 MHz	4 GB DDR3, 1066 MHz	8 GB DDR3, 1600 MHz
OS	Win8.1 x64	Win8.1 x64	Win7 x64

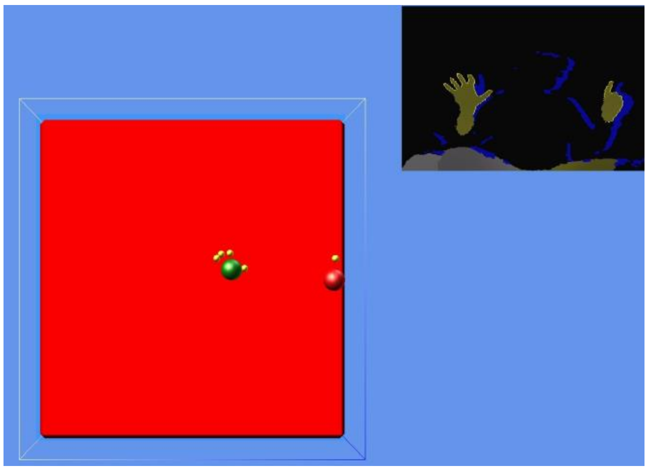

Figure 4. Example of the used benchmark scenario where simple cube was loaded while finger recognition was active

that the best distance is if the user is positioned at around 1.2m. Another case is when Kinect confuses other parts of environment or other joints as hands (usually when a user moves too fast) in which case wrong image fragment is used when trying to detect fingers. In general, most problems happen because Kinect does not have high enough temporal and spatial resolution and all of these problems should disappear in future implementations since use of other controllers is already planned.

Since there are no directly comparable products on the market which offer gesture based modelling, we subjectively evaluated the usability and satisfaction of users after using our prototype application. Although our proposed solution, in its current state, lacks the precision of the professional 3D modelling tools, feedback collected from the users proves that the concept of intuitive, computer based, 3D modelling is feasible. Simplicity and ease of use of the developed prototype application became apparent when users without any modelling experience managed to quickly figure out the gestures and started modifying the preloaded model.

TABLE II. PERFORMANCE DEPENDING ON THE MODEL SIZE

Cube length	Configuration 1	Configuration 2	Configuration 3
(N vertices)		*(FPS)*	
50	127	358	670
100	24	57	124
150	7	20	42
200	4	8	18
250	2	4	10

V. FUTURE WORK

As an interesting future research topic and potential improvement, we consider changing the controller. Since Kinect was designed to recognize larger body parts, competitive devices with finer recognition capabilities would be a better choice for this application, such as Leap Motion Controller or the new Kinect v2. Another interesting feature, which would improve immersion and usability, would be an interactive perspective projection transformation based on the user's head position which is easily obtainable through Kinect API. The idea is that the virtual view responds to the user's head movement, and we believe this feature would positively affect the experience of the virtual 3D world. There is also high probability that the mentioned feature would be also implemented with a virtual headset, such as Oculus Rift, since it has higher availability compared to other similar devices.

There is also some room for improvement regarding the application's performance. One of the optimizations is related to eliminating the redundant vertices created by the marching cubes algorithm since the edges on the neighbouring cubes share the same vertices. The idea would be to create only a single instance of each vertex and then use indexing to describe where each vertex is located. Another useful optimization would be partial dynamic model loading, especially when dealing with high resolution volume models. Since the complexity of the algorithm is high, this feature could considerably

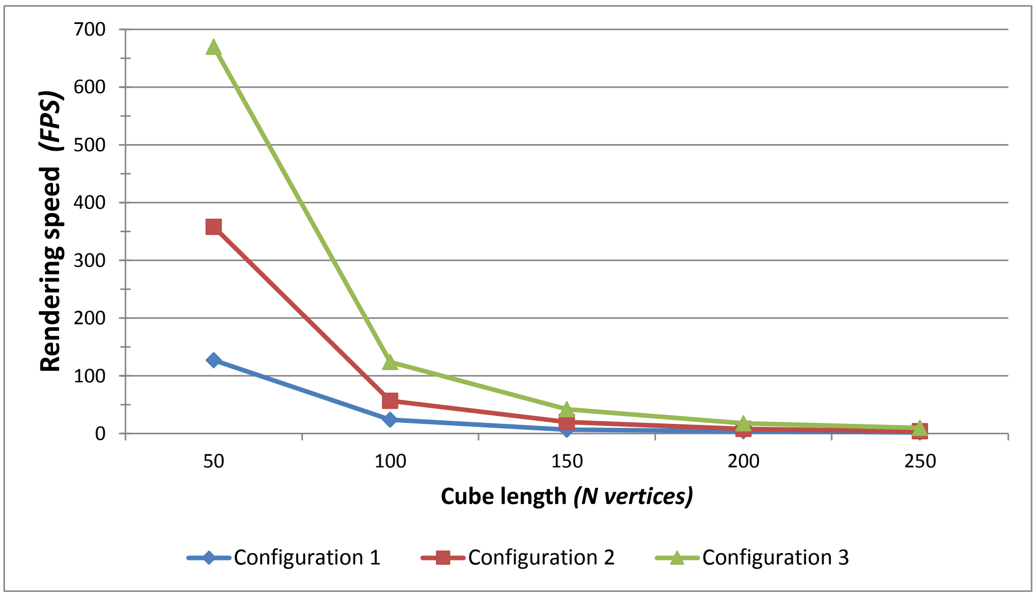

Figure 5. Graphical representation of the benchmark results

affect the performance of the program when used on large models. An implementation would probably be similar to the culling technique where a model could be divided into smaller regions and when comparing depths those that are further from the camera would not be processed.

VI. Conclusion

Natural user interface technologies can be more than just a platform for gaming and entertainment because they possess a great potential in revolutionizing human-machine interactions. In these types of interface designs 3D motion sensing input devices like Microsoft Kinect have proven to be a suitable and valuable hardware choice. However, the effectiveness of the hardware heavily depends on the accompanying software which still does not follow any generally accepted interaction model. This article proposes a solution to this problem in the form of a functional prototype that tries to imitate a real life experience of traditional human skills such as clay modelling.

A functional prototype presented here forms the basis for the future versions with an improved interaction system, which could provide tremendous opportunities for addressing various problems, and also opens new frontiers and new challenges in user interaction with virtual objects.

References

All web links have been tested and functional on February 8, 2015.

[1] Oliver Kreylos' Homepage, Wiimote Hacking. http://idav.ucdavis.edu/~okreylos/ResDev/Wiimote/index.html (posted October 2007)

[2] A developer's perspective on immersive 3D computer graphics. Virtual clay modeling with 3D input devices. http://doc-ok.org/?p=493 (posted March 2013)

[3] Cho, S., Heo, Y., & Bang, H. (2012, August). Turn: a virtual pottery by real spinning wheel. In ACM SIGGRAPH 2012 Posters (p. 11). ACM.

[4] Cho, S., Baek, D., Baek, S. Y., Lee, K., & Bang, H. (2014). Three-dimensional Volume Drawing on a Potter's Wheel. IEEE Computer Graphics and Applications, (1), 1.

[5] Dave, D., Chowriappa, A., & Kesavadas, T. (2013). Gesture interface for 3d cad modeling using kinect. Computer-Aided Design and Applications, 10(4), 663-669.

[6] Ryan, D. J. (2012). Finger and gesture recognition with microsoft kinect. http://brage.bibsys.no/xmlui/bitstream/handle/11250/181783/Ryan%2c%20Daniel%20James.pdf?sequence=1

[7] Lorensen, W. E., & Cline, H. E. (1987, August). Marching cubes: A high resolution 3D surface construction algorithm. In ACM siggraph computer graphics (Vol. 21, No. 4, pp. 163-169). ACM.

[8] Geiss, R. Generating Complex Procedural Terrains Using the GPU. In GPU Gems 3; Nguyuen, H., Ed.; Addison Wesley Professional, USA, 2008; pp 7-39.