Shaka Laka Boom Boom: 2D Cartoon Sketches to 3D Models

DIVYANSHU TALWAR, IIIT Delhi
ARPIT BHATIA, IIIT Delhi
SHUBHANG SATI, IIIT Delhi

In this work we aim to design a system to create a scene of 3D objects synthesized from 2D paper drawings and sketches which can be moved around (rotated, scaled and translated) interactively using hand gestures with the help of Leap Motion. When working on this work we were reminded of the magic pencil in the fictional show `Shaka Laka Boom Boom` we used to watch as kids. This magic pencil brought to life anything which was drawn with it which aligns perfectly with this project, hence the name.

1 LITERATURE SURVEY

In their paper, Feng et al. summarize the previous work done in the fields of AR books, sketch-based modelling systems and AR authoring models. We discuss these works in brief in the following subsections.

1.1 AR Books

Researchers have tried to combine augmented reality and books to enhance the learning experience for children and make it more entertaining. Some books such as the Magicbook[3] render virtual characters on top of the pages of a book for the purpose of storytelling. Another example is AR coloring books which consist of pre-defined models corresponding to each coloring figure. The texture of this model is changed based on how the corresponding figure is coloured. Examples of such technology include the work done by Magnenat et al. [4] and Crayola Color Alive [5]. The drawback of an AR coloring book is that a user cannot created their own models and only rely on those provided by the manufacturers of the books.

1.2 Sketch-based Modeling System

Creation of 3D models from sketches is an area which has been well explored through works such as FibreMesh [9] and ShapeShop [6]. However, current systems are limited to the creation of models having spherical topology. These systems also require a large number of sketches corresponding to different views, which makes the process cumbersome and unattractive to children. This work aims to address these issues by using a distance field based modelling technique which requires a single sketch and is free from having the constraint of spherical topology.

1.3 Authoring Models in AR

Authoring tools are used to create 3D models and scenes in an AR. Tools such as ARpm [7] use methods similar to traditional 3D modelling tools and are thus not suitable for children due to their complex nature. Other tools such as the one presented by Hagbi et al. [8] allow for the creation of AR scenes by superimposing pre-created virtual objects on pre-defined types of sketch. This approach faces the same drawback as AR coloring books, inability to create custom models. This project provides a workflow for creation of AR scenes by automatically generating 3D models based on sketches and enables the user to place the models though leap motion based interactions.

Authors' addresses: Divyanshu Talwar, IIIT Delhi, divyanshu15028@iiitd.ac.in; Arpit Bhatia, IIIT Delhi, arpit16229@iiitd.ac.in; Shubhang Sati, IIIT Delhi, shubhang16198@iiitd.ac.in.

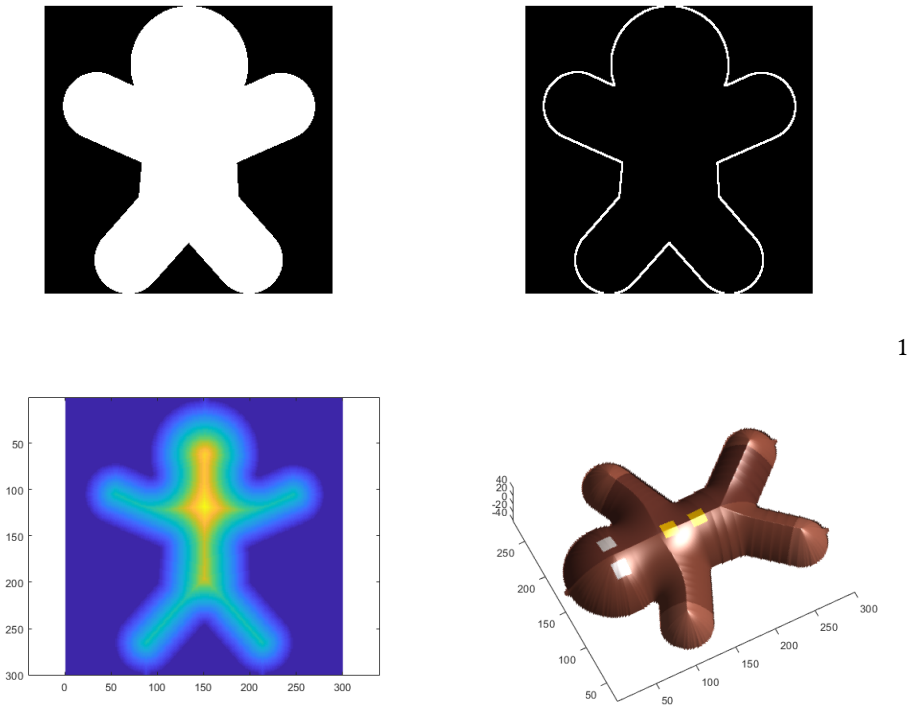
2 ALGORITHMS AND IMPLEMENTATION DETAILS

2.1 Image Processing

The first step involves resizing the image to a fixed height of 300px while preserving the aspect ratio of the image captured. This is ensures that model sizes are consistent across a scene and reduces the complexity of the mesh. We then proceed to extracting the outline of the image. We first convert the image from RGB color space to HSV color space and apply further image processing steps on the saturation channel. This is done as the saturation channel has the most sensitive visual content. To this, morphology close and open operations are applied followed by a dilation operation to fill up small gaps in the image and fetch only the boundary ignoring the outlines of elements contained within it such as eyes. Based on the boundary obtained, we use a flood fill operation to get use the region maps of the image. This helps us differentiate between the outside of the image (which we ignore while creating a depth map) and the inside (which ultimately forms our mesh).

2.2 Inflation

The first step for mesh generation is to create a euclidean distance transform of the obtained outline. For the areas outside the outline, we set the distance transform to zero. The distance values obtained currently have a linear variation and will result in a very sharp model if directly used. Thus the following circular smoothing



1

Fig. 1. The images from left to right, top to bottom are the Region Map, Image Boundary, Depth Map and Final Result respectively.

function is used:

$$H(x) = \sqrt{D_{max}^2 - a^2}$$
$$a = D_{max} - D(x)$$

where $D(x)$ is the distance value of the current vertex and D_{max} the maximum distance value within this region. We plot this generated map twice as a point cloud surface to get the required mesh - once with their actual height values and next with height values as the additive inverse of the actual height used earlier. A MATLAB surface (comprising of two stitched sub-surfaces) is then elevated using the generated depth map to view the intermediate results. This surface is then texture mapped using our script which warps a 2D sketch image on top of it to create a texture mapped 3D MATLAB surface.

The output of this step is a texture mapped MATLAB surface plot which is later used for mesh generation and is exported as an `obj` to the mesh via the later described server-client setup.

We have also added an extra component over the referred paper of by removing the bottom 30% height values. This is done to obtain a better join when combining the surface plot to get front and back faces of the model.

2.3 Mesh and obj generation

The texture mapped MATLAB surface plot obtained after the inflation step is then used to extract the vertex wise color values (since, the texture mapping was done in MATLAB using a custom script).

We then convert the point cloud (structured) generated in the previous step into a mesh by selecting a set of four points forming a quad and then representing it as a set of triangles which are then simultaneously stored into the `obj` file - using our custom `obj` wrapping script. The final `obj` file also contains the vertex-wise colour values i.e. a bit more as compared to the contents of a standard `obj`. We later, in Unity write our own un-wrapping script to display these textured 3D-models.

2.4 Server Client Model

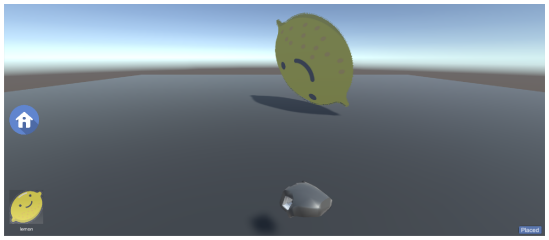
Since the inflation and mesh generation tasks are computationally heavy and can not be run on the client side always, we have made the system modular by separating the mesh generation and rendering parts. Mesh generation is done using MATLAB on a remote machine and the rendering will be done on a completely isolated machine (connected via internet). We here use two systems to demonstrate the aforementioned with of them acting as the server and the other one rendering the generated 3D model on Unity Engine.

2.5 LeapMotion

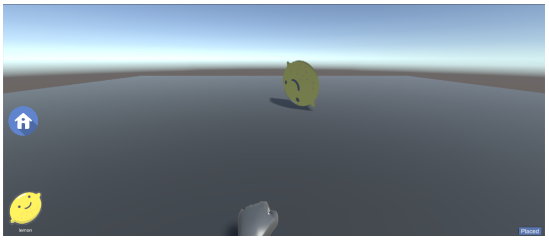
We used a Leap Motion sensor to provide the functionality of using hand gestures to create a scene containing our generated 3D models. We have created different hand gestures corresponding to different transformations we can apply to the mesh. The gestures are implemented using the hand properties and detection utilities available in the Leap Motion Unity Package. Each gesture had to have a trigger so that the user can choose the transformation they want to apply directly using their hands, without having to use the GUI. The gestures we implemented as listed below:

- Scale: The left hand when clenched to a fist is used to scale the object based on the distance of the fist from the Leap Motion.
- Rotate: The right hand when clenched to a fist is used to rotate the object based on how the wrist is being rotated.

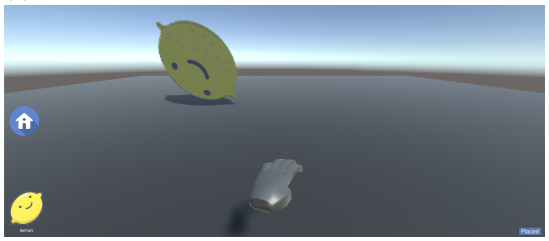
- Translate(Along Z axis): When both hands come into the sensing area of the Leap Motion, the object is moved forward or backward based on the direction both palms are facing.
- Translate(Along X axis): When the four fingers are extended out while the thumb is kept closed, we trigger movement along the x axis. The thumb of the hand is used to move an object right while the left thumb moves an object left.



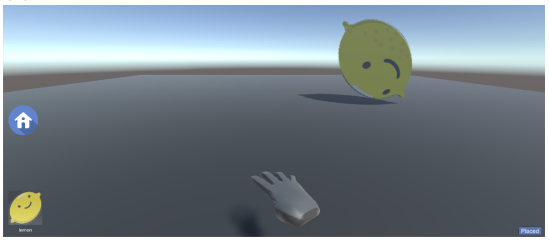
(a) Rotate Gesture



(b) Scale Gesture

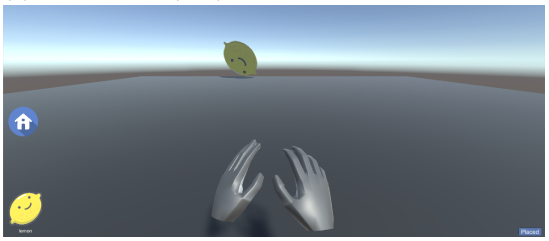


(c) Translate X (-ve) Gesture



(d) Translate X (+ve) Gesture

(e) Translate X (-ve) Gesture



(f) Translate X (+ve) Gesture

2.6 The pipeline

The application opens on the client side, having three options, namely generate a 3D object (from a 2D image), create a scene (from the uploaded objects) and settings menu - where in the user can set the address of the server (responsible for inflation and mesh generation).

- **Generate a 3D object**: In this step the user can upload an image (here `.png`) - using the underlying server-client setup (figure 2). A watcher script watching the upload destination (on the server side) then triggers the response of launching the `inflator` MATLAB function (described above). This then returns an `obj` file for the image in question in a dedicated directory on the server. As soon as this is done, the client-side - polling for this file since image upload - fetches it and adds it to the available 3D models list. All this happens in real-time.

- **Create a scene**: Here the user can create a scene (figure 3) using the 3D objects generated from the uploaded 2D images. One can add a 3D object to the scene by simply clicking on it's respective placeholder displayed in a list placed at the bottom of the screen. This event triggers a response running the script

responsible to un-wrap the required `obj` file displaying a textured 3D object in the scene. This object can now be translated, rotated and scaled by the user who can freeze it's location anytime they please by pressing the "`placed`" button (placed at the bottom right of the screen). User can place as many objects as they like, and can even return to the main menu, if they wish to populate the 3D object script further. On completion, one can choose to click a screen-shot of their work, capturing their creativity (on paper) as a 3D scene.

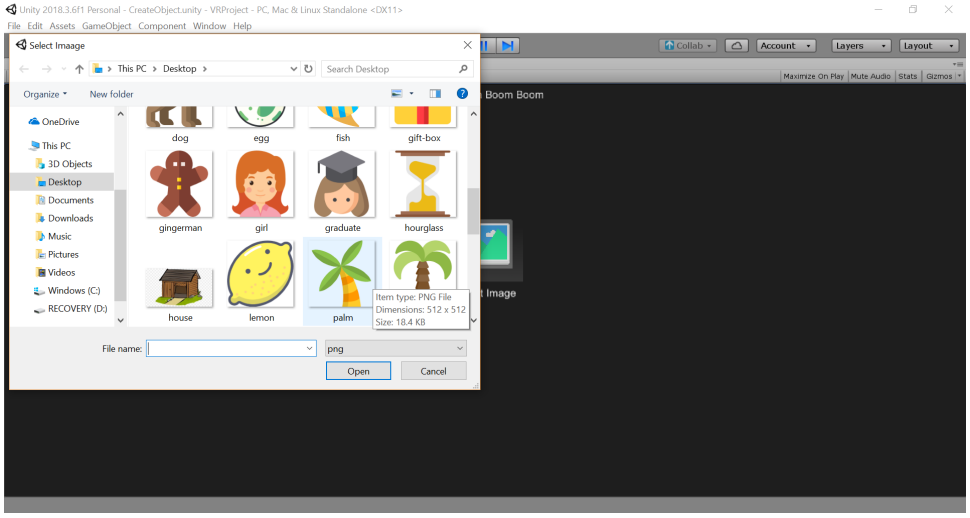


Fig. 2. Generate a 3D object - Image upload screen.

Fig. 3. Final output of the 'create a scene' step.

3 CHALLENGES FACED

(1) Exporting the MATLAB surface (generated after the inflation step) as an `obj` file which could later be imported into Unity.
- To solve this we wrote custom `obj` wrapper and un-wrapper scripts on MATLAB and Unity respectively. The texture mapping done in MATLAB is also exported in this `obj` file as it stores vertex colours too. The implementation specific details for the same can be found in section 2, subsection 2.3.

(2) Importing correct normals for the seam-region into Unity.
- Unity when displaying a 3D object, ignores the vertex normal information provided in the `obj` file. Instead, it calculates them using the vertex orderings for a given face. Due to this, we had to come up with a hack to minimize the incorrect-normal calculation in the boundary or the seam region. The hack greedily decided to make the vertex ordering of a face's constituent vertices consistent with the side containing majority of its constituent vertices (i.e. if 2 of the vertices lie on the side with faces having a particular vertex ordering, this face would have a vertex ordering similar to it).

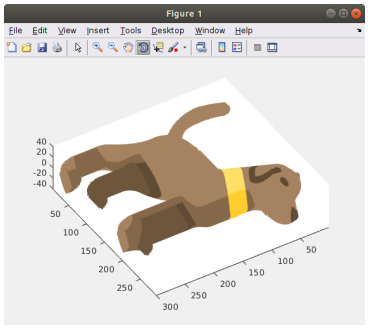
(3) Leap Motion Integration.
- We faced some issues implementing the Leap Motion based gestures due to outdated documentation and lack of tutorials. Quite a few of the functions mentioned in the docs throw errors with the latest Unity package.
- There are also useful functions for hand tracking which are not mentioned in the docs. For eg. we wasted some time trying to get the wrist rotation based on the palm normal but eventually discovered there was already a variable storing this information, except this variable was not mentioned in the docs.
- We were able to implement what we needed to but it took us some time to get the hang of how to use the Leap Motion Unity Package.

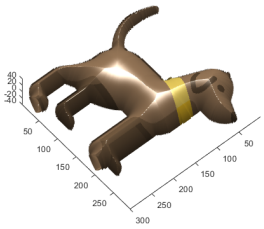
4 STATUS OF DELIVERABLES

4.1 All Completed

(1) Image processing to generate region and depth maps.
(2) Generating the 3D model using the above mentioned results.
(3) Server-client channel.
(4) Loading the model(s) on the app.
(5) Leap Motion hand gesture detection.
(6) Integrating detected hand gestures with the app to interact with the models.

5 RUNNING THE ALGORITHM ON DIFFERENT IMAGE SAMPLES





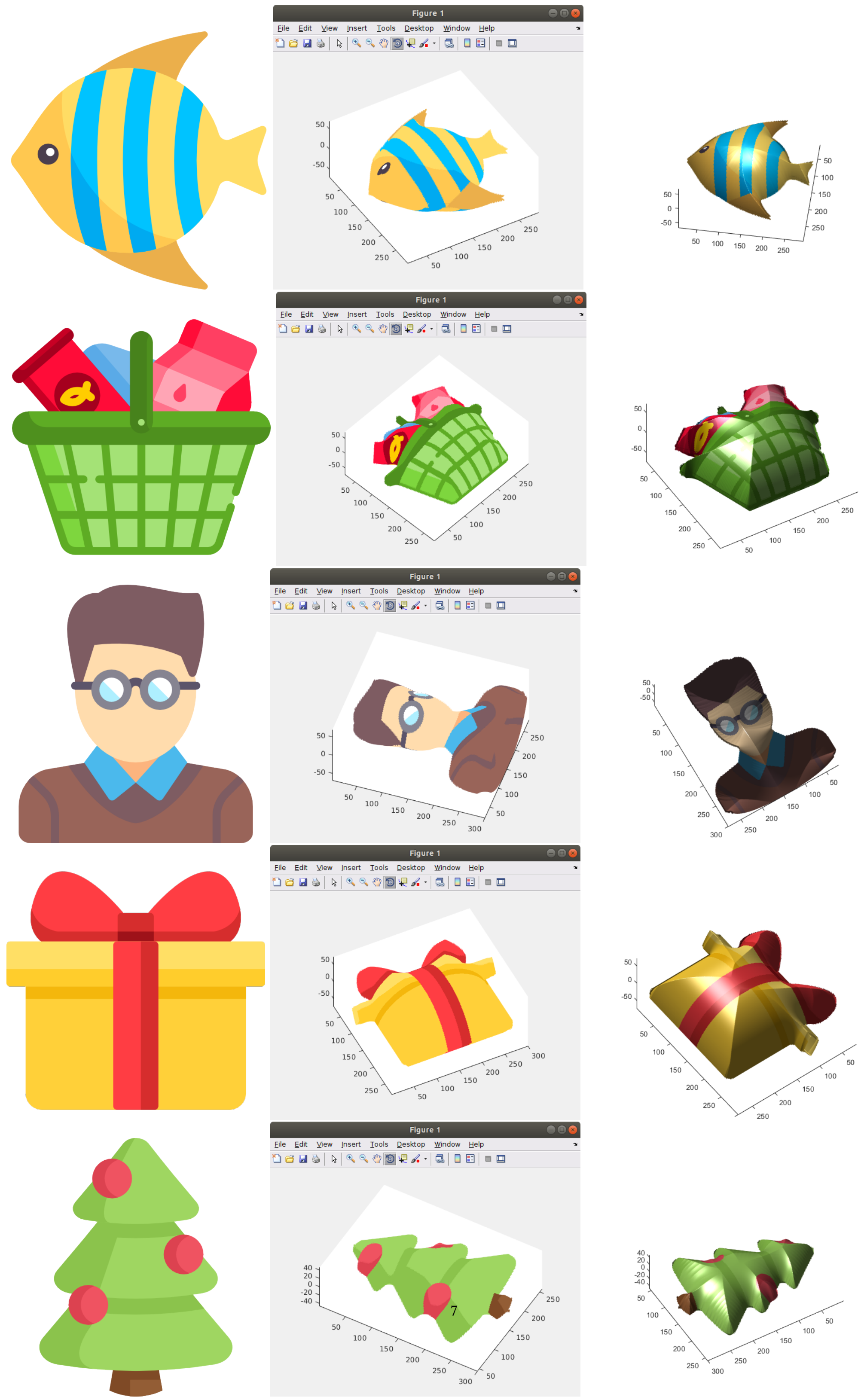


Figure 1
File Edit View Insert Tools Desktop Window Help

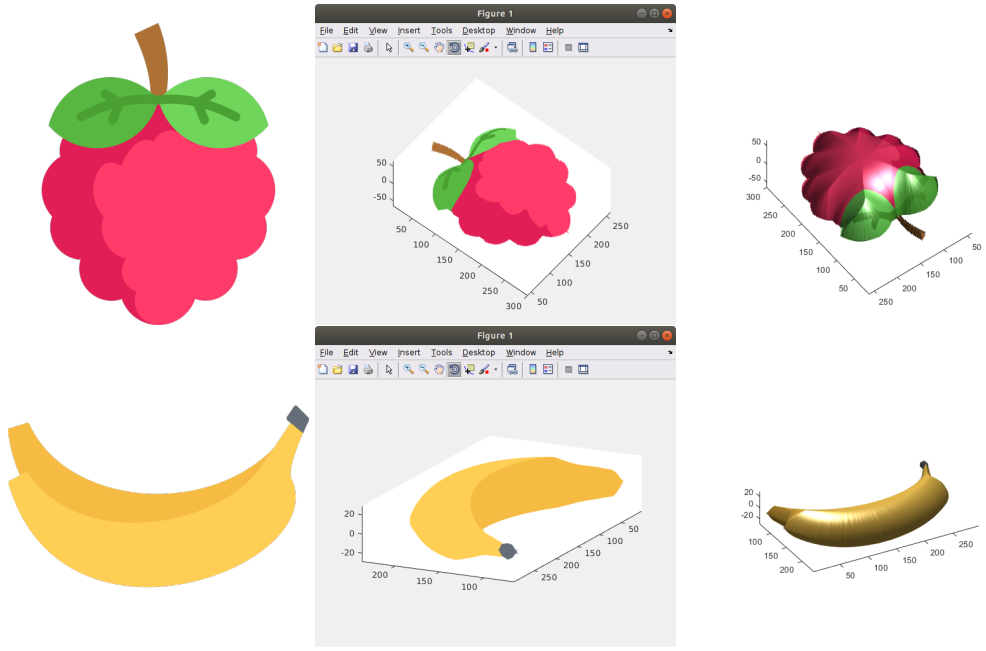


Fig. 4. Results for inflation, where the first image (on the left) is input - 2D sketch and the images towards its right are the corresponding inflated 3D models.

REFERENCES

[1] Feng, Lele an Yang, Xubo and Xiao, Shuangjiu (2017) “MagicToon: A 2D-to-3D Creative Cartoon Modeling System with Mobile AR". IEEE Virtual Reality.

[2] L. Olsen, F. Samavati and J. Jorge (2011) “NaturaSketch: Modeling from Images and Natural Sketches" IEEE Computer Graphics and Applications. vol. 31, no. 6, pp. 24-34.

[3] M. Billinghurst, H. Kato, and I. Poupyrev. Magicbook: Transitioning between reality and virtuality. In CHI âĂŹ01 Extended Abstracts on Human Factors in Computing Systems, CHI EA âĂŹ01, 2001.

[4] S. Magnenat, D. T. Ngo, F. Zund, M. Ryffel, G. Noris, G. Rothlin, A. Marra, M. Nitti, P. Fua, M. Gross, et al. Live texturing of augmented reality characters from colored drawings. Visualization and Computer Graphics, IEEE Transactions on, 21(11), 2015.

[5] Crayola. Crayola color alive. https://www.crayola.com/splash/products/ColorAlive

[6] R. Schmidt, B. Wyvill, M. C. Sousa, and J. A. Jorge. Shapeshop: Sketch-based solid modeling with blobtrees. In ACM SIGGRAPH 2007 Courses, SIGGRAPH âĂŹ07, 2007.

[7] P. Fiala and N. Adamo-Villani. Arpm: an augmented reality interface for polygonal modeling. In Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on, 2005.

[8] N. Hagbi, R. Grasset, O. Bergig, M. Billinghurst, and J. El-Sana. Inplace sketching for content authoring in augmented reality games. In Virtual Reality Conference (VR), 2010 IEEE, 2010.

[9] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: Designing freeform surfaces with 3d curves. In ACM SIGGRAPH 2007 Papers, SIGGRAPH âĂŹ07, 2007.

[10] Stoyan Kerefeyn, Stoyan Maleshkov (2015) “Manipulation of virtual objects through a LeapMotion optical sensor". IJCSI.