Revista Romana de Interactiune Om-Calculator 10 (2) 2017, 119-146

Augmented Space Editor

Codrin Dumitru Goia[1], Dorian Gorgan[2]

Technical University Cluj-Napoca
Memorandumului 28, Cluj-Napoca
[1] *E-mail: codrin.goia@gmail.com*
[2] *E-mail: dorian.gorgan@cs.utcluj.ro*

Abstract. The paper presents a new concept of human-computer interaction having as purpose 3D object modeling. The user models an object in an augmented reality context, using as input the touch screen and two markers, the first representing the scene base on top of which shapes will be drawn, and the second being a 3D cursor. The research experiments a robust camera pose estimation algorithm based on the detection of the marker, allowing for pose retrieval even in situations where marker is partially occluded.

Keywords: augmented reality, 3D editor, 3D cursor.

1. Introduction

As hardware and software evolve more and more we observe, among other phenomena, a continuous strive to find new methods of interaction between human and computer. Human-computer interaction is an important field of research since it focuses on the interface between two inherently different entities, the person (user) and the computer. Communication plays an essential role as each of the two ends must present its information in a form known to the other. A solution that targets this problem is augmented reality.

Augmented reality (AR) is a live direct or indirect view of a physical, real-world environment whose elements are augmented by computer-generated sensory input such as sound, video, graphics or GPS data. It is related to a more general concept called computer-mediated reality, in which a view of reality is modified (possibly even diminished rather than augmented) by a computer. Augmented reality enhances one's current perception of reality, whereas in contrast, virtual reality replaces the real world with a simulated one. Augmentation techniques are typically performed in real-time real time and in semantic context with environmental elements, such as overlaying supplemental information like scores over a

live video feed of a sporting event.

In other words, augmented reality represents a new paradigm of interaction with computers by linking real and digital world together. Rather than needing to fight his way through abstract representations, complex menus or textual representation regarding a specific task, the user can now interact intuitively with objects from his own world, in real-time (Wendy, 1998). Furthermore, the trend in AR is upwards, signalizing the positive experience on the user side and great yet incompletely exploited potential of this technology (Buchholz, 2014).

In computer graphics, 3D modeling is the process of developing a mathematical representation of any surface of an object in three dimensions via specialized software. The product is then called a 3D model. Once created, it can then be displayed as a two-dimensional image through a process called 3D rendering, it can be used in computer simulation of physical phenomena, can be physically created using 3D printing devices, or be further used in other applications such as animation software, video post-production, or games.

When the user is interacting with the computer to work on a 3D model, ambiguity is introduced due to the two-dimensional nature of the monitor. Ideally, the object being modeled is both in the designer's mind as well in the computer memory, in different representations. However, the communication between these two instances of the same object is limited due to the fact that every interaction is done through a two-dimensional image on the monitor. Due to this fact, regardless which angle is chosen to look at the model in the virtual scene, or which lightning model is used, one dimension is always lost in the process of communicating the model.

Consider the following two example scenarios:

1. In the virtual scene, the camera is focused on the object and the user desires to move a part of the object being modeled towards the camera. Because the camera is facing the object and the image formed on the camera is displayed on the monitor, the task is equivalent to moving the object out of the monitor. Hoverer this represents a new dimension which the screen is not able to reproduce, hence the user cannot point to it.
2. There is a fixed point selected in the virtual scene. The camera used in the virtual scene is facing the point, such that it becomes visible on the monitor. However, the user is unable to deduce distance of the point

to the camera, because the point can be anywhere on the line connecting the position of the point and the center of projection of the camera.

Both examples show the ambiguity that arise when dealing with a three-dimensional, virtual scene. These obstacles can be overcome in each case by changing the angle from which the camera is looking at the virtual object, such that the position in 3D space becomes well determined. However, this requires extra work and constant attention on behalf of the user.

Because of this and similar reasons, 3D modeling systems are tedious to use and generally shackle creativity of users (Stéphane et al., 2003).

This paper proposes a new approach to modeling 3D structures by introducing augmented reality interaction in the context of a 3D modeling software. The user (3D artist) is able to define shapes in the real world by using a custom defined pencil in his one hand, and view the created shapes in real-time in the context of an AR scene on a mobile device, held in his other hand.

The paper is structured as follows. Next section …

2. Related Works

There are a number of fields related to augmented reality. Augmented reality is actually part of a bigger picture, namely the concept of mixed reality (Schmalstieg and Hollerer, 2016). Mixed reality, sometimes referred to as hybrid reality, is the merging of real and virtual worlds to produce new environments and visualizations where physical and digital objects co-exist and interact in real time. Augmented reality contains primarily real elements and, therefore, is closer to reality. For example, a user with an AR app on a smartphone will continue perceiving the real world in the normal way, but with some additional elements presented on the smartphone.

The first operation in a long AR pipeline process is to detect the markers representing the scene and the user pencil, before passing their data on and proceed to their full analysis to get 3D poses. As the detection process is color-based, each pixel in each frame from the camera feed is analyzed and compared against the value that it searches for.

A widely used color model is the RGB color model. While using this color model has its advantages, it has the limitation that it is heavily dependent on illumination transformations. As such, two similar colors

illuminated at different light intensities may look equally different as two different colors that have the same brightness. This represents a limitation when searching for a specific color in a camera shot, as the interest lies in the originating source color and not in the result appearing on the camera, result that may be affected in brightness by different light sources and shadings. This limitation yields to the HSV color model (Szeliski, 2010).

Siltanen describes in (Siltanen, 2012) the process to define the camera in marker coordinates. This is a matrix based camera transformation that localizes the camera with respect to the marker, and it stays the same for one frame. This is a non-linear estimation problem and will be solved using a least-square minimization algorithm, such as the Levenberg-Marquardt algorithm (Lourakis, 2005).

In normal AR scenarios the camera pose is computing by analyzing a fully detected fiducial marker in the form of a quadrangle. However, in practice one corner or even one full edge of the quadrangle may be occluded leading to a situation where camera pose extraction is possible in theory but requires a whole new approach to be put in practice. The paper of (Alvarez and Borro, 2009) proposes an algorithm to address this issue by computing the camera pose the normal way when the marker is fully visible and then, when the marker gets partially occluded, making estimations based on the poses of the previous frames.

3. Augmented 3D Space Editor Solution

3.1 Conceptual Architecture of the Editor

The application features a new approach on interacting with virtual 3D structures with the aim to be more usable and intuitive. Values such as ease of use and natural ways of human-computer interaction are put above precision and complete control. In other words, the application will enable users with minimal knowledge in 3D modeling to express their ideas directly, without the need to master the complex concepts that come with this field.

Figure 1 shows how the data flow bridges the gap between input and output while communicating with and updating the model. The entire workflow is heavily optimized to work in real-time.

There are two inputs to the system, the camera and the touchscreen.

Different from the camera input, the touch screen input is only triggered when the user operates on the user interface. Data is modified in the model accordingly, and will be taken into consideration while processing future camera frames. The input arriving from the camera is different. It represents a continuous loop and feeds the application on a regular basis, that is for each frame (there are desirably at least 24 frames in one second); new input arrives with every camera frame, which triggers its processing.

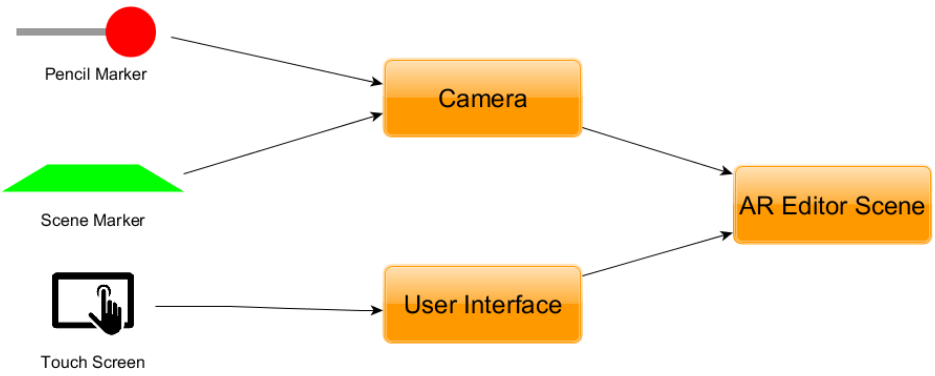

Figure 1. Conceptual architecture of the Augmented Space Editor

3.2 Core Features Analysis

During the main use case scenario, the user will be editing virtual shapes in an interactive augmented reality space. This requires the application to do the following high-level tasks at each update:

1. Extract the camera pose from which the current frame was taken;
2. Compute the position of the user pencil and match it to the context of the scene;
3. Modify the model in accordance with the control commands coming from the user interface and the computed pencil position;
4. Redraw the scene in conformity with the pose of the new frame and the updated model.

These coarse tasks will now be grouped in input and output related tasks, split in subtasks and each of them be considered individually from a theoretical perspective.

4. Input Features

There are three types of input features the processing concerns with: scene marker, pencil position, and user interface.

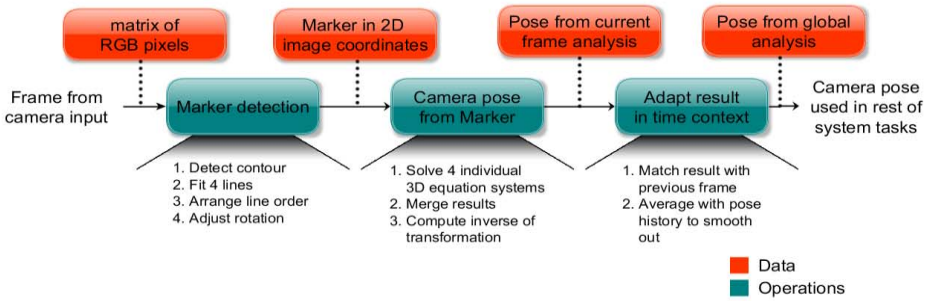

Figure 2. Operations pipeline to get from input frame to camera pose

4.1 Camera Pose

The extraction of the camera pose is the core task that stands at the base of all the other tasks. Computing it for a given frame means to compute the position and orientation of the camera that captured that frame, with respect to the scene marker that is identified in it. This will enable us to capture the virtual scene from the exactly same position and orientation with respect to the scene base, and then add the result of this capture over the original frame. In the eyes of the user, this will make the model in the virtual scene look as part of the real world, because two perspectives from which we are looking at the two scenes (the virtual and the marker) are one and the same.

Our implementation of camera pose retrieval is different from other implementations on this matter. It is designed to compute the camera pose even in situations where the scene marker often gets partially obstructed by the user pencil marker and is not fully enclosed in the camera screen. This makes the algorithm very flexible and robust to user input.

Figure 2 shows the pipeline of operations that the input frame undergoes to obtain the camera pose, whose result will be used further on in the application. The pipeline process is executed for every frame coming from the camera.

Scene Marker Type

We will now describe the marker that the system scans for in order to obtain the camera pose. This marker represents the scene base to the user, on top of which he will model the virtual shapes. This marker will be further referred to as the scene marker.

An important aim of the project from start was to require the user a minimal amount tools in order to use this application. That imposed the system to accept a marker that can be easily provided from his environment without asking him to make measurements with the ruler or print out special markers. The scene marker represents a square piece of paper of custom color.

Marker Detection

The detection of the scene marker is the first stage in the pipeline for determining the camera pose. As input it receives the current frame from the camera and the color attributes of the marker to be tracked, and at the end of this stage it outputs the marker detected in the frame, in 2D image coordinates. This output will be further processed by the next algorithms in the pipeline to extract the 3D camera pose.

The marker in the real world is a square piece of paper; on the frame, due to its projection on the camera screen, it forms a convex quadrangle. At this stage all operations are in 2D space. It consists of the following tasks:

1. Contour Detection. Detect the contour of the marker;
2. Line fitting. Fit four lines in the extracted contour to obtain the marker edges;
3. Rearrange edges. Rearrange the order in which the lines are considered such that it determines the edges of the marker;
4. Adjust rotation. Rearrange the order in which the lines are considered, such that they are placed in counter clockwise order.

Contour Detection

In order to extract the contour, we first scan the image in the search for regions that have the color we are looking for. Because the color of the marker in the environment may appear on the camera brighter or dimmer, depending on the current illumination; we rather want a comparison independent on the environmental light which will enable us to take for the comparison the originating color of the object and not its appearance on the camera screen. In order to achieve this, we convert the color model from

RGB to HSV and then perform the comparison between each image pixel and the color we search for within the given radius, independent on illumination. This conversion is done for each pixel individually and is a basic operation consisting of a few additions, multiplications and divisions.

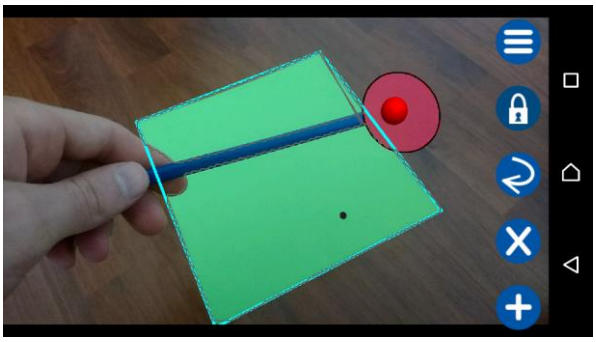

Figure 3. Marker detected correctly (light blue) even if it is formed out of two non-adjacent groups, above and below the pencil (contour of the groups shown in gray)

The algorithm (Figure 3) is able to process as well the cases when the pencil marker in the hands of the user might be placed in the image in such a way that it is in front of the marker, partially covering it in the middle and splitting it in two parts that do not touch each other.

Rearrange Edges

Although we now have all the marker edges expressed as lines in the 2D image, it is not enough information for the pose estimation algorithm. At this point, data is a random line sequence. We need to order them in such a way that their traversal from line 1 to line 4 represents a quadrangle.

Line Fitting

We now have a contour (a closed chain of pixels) of which we now it represents a square marker. Hence, we will successively apply 4 times a Ransac line fitting algorithm, and after each iteration we will remove the pixels that supported the current resulting hypothesis. That means the first time we apply Ransac on all the points. After finding a line, all the points that are in the proximity of that line are removed from the original set of points, so that they will not be considered again in future iterations – after all we found the line and don't want it to interfere with future searches of other lines. We then apply Ransac again in search of the second line, on the

set of points that remain, and so on.

Ransac algorithm for line fitting is very fast. What it does is to randomly select two points out of the set of points and build the line that connects them. Next, the set of points is considered again to check how many points actually support this line hypothesis, that means how many lie in its proximity. The line and number of its supporters is remembered and the same process is repeated a few times. At the end, the line with the most supporters is returned.

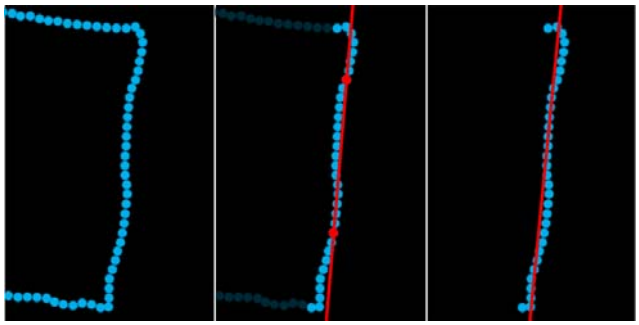

Figure 4. From left to right: Original contour points, line result after Ransac fitting, revised line result after Least Squares fitting on Ransac supporters

To maximize accuracy, after fitting a line with Ransac, the line is re-estimated using a least squares algorithm performed on its supporters. The line now represents the best possible estimate of the original points (Figure 4).

Adjust Rotation

The final step in the algorithm for identifying the 2D marker in the image is to ensure the iteration through lines l_1-l_2-l_3-l_4 is done in anti-clockwise direction. This is done using the rotation matrix in the 2-dimensional space, which defines a rotation around the origin by an angle α defined in radians.

Camera Pose from Marker

The input of this pipeline stage is the output from the previous stage which represents a marker defined in the 2D coordinates of the frame with the following characteristics:

- It is formed out of the line sequence l1-l2-l3 or l1-l2-l3-l4, which represents the 3 or 4 detected marker edges;

- The line sequence is ordered;
- The line sequence goes counter-clockwise.

At this point the algorithm leaves the 2D space of the frame behind and enters the 3D world. What it does is to analyze the 2D input data in order to obtain its significance in the 3D context. It consists of the following tasks:

1. Solve the individual 3D equation systems
2. Inverse the transformation
3. Merge results

Solve the Individual 3D Equation Systems

When filming with a camera, objects from the real world get projected on the camera screen where they are represented by color pixels. During this process ambiguity is introduced, as the depth information is lost. As an example, a point in the real world when filmed by a camera will get a pixel value on the camera screen that corresponds to its perspective projection on the camera screen. However, it is represented only as a color value and the information regarding the distance which the original point has to the camera is lost. That means by simply analyzing a pixel in the image we cannot determine the location of the point in the outside 3D world that caused that projection. The point could be anywhere on the line connecting the camera center of projection and its pixel representation on the camera screen.

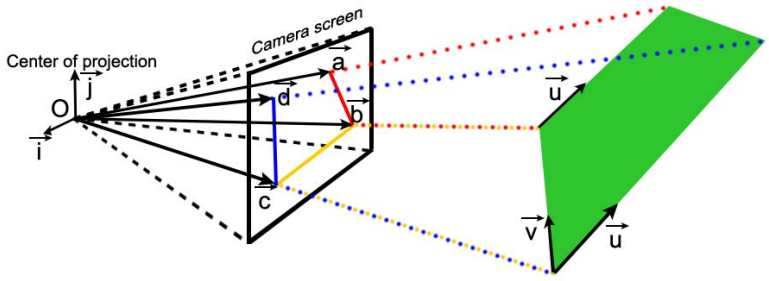

Figure 5. Transformation to camera coordinates system

The proposed algorithm that computes the pose does not need more than 3 marker lines as input. This means it is necessary and sufficient to detect 3 lines of the marker in order to get a pose estimation. We aim to express the

3D pose of the marker identified in the frame with respect to the camera coordinate system. The marker has the form of a square.

Let us now consider any two adjacent edges of this square (Figure 5). As these edges are perpendicular and intersect each other, we can consider one as being the marker's ox axis and the other one the marker's oy axis. We denote the marker's ox axis by $\vec{u}$, the marker's oy axis by $\vec{v}$, and the marker's position by $\vec{p}$. With these notations the marker is completely described. Thus, we say that we found the pose of the square/marker with respect to the camera if and only if we found $\vec{u}$, $\vec{v}$ and $\vec{p}$. These are all vectors in 3D space. $\vec{u}$ and $\vec{v}$ completely define the marker's orientation, while $\vec{p}$ defines its position in space. After we find out their values we will switch back to coordinate system of the marker. Thus the orientation of the marker is computed extremely fast, by applying some 3D vector space geometry.

For determining the marker position, we leave the orientation phase where we dealt only with angles and introduce distance data to our system. As such, we use the base for our distance computations the measured constant length of one side of the scene marker (i.e. 290 mm).

Therefore, we have computed the orientation (i.e. $\vec{u}$, $\vec{v}$) and position (i.e. $\vec{p}$) of the marker in camera coordinates, in other words the marker pose/transform with respect to the camera.

Inverse Transformation

The camera pose with respect to the marker is simply obtained by computing the inverse of the transform. This represents a change in perspective in the algorithm. Until now all data was expressed taking the camera as a base. This was necessary because the camera was the only thing that we knew in the environment.

Merge Results

As mentioned earlier, the resulted pose was computed taking into consideration only 3 marker edges. Should the current frame contain a partially visible marker with only 3 visible edges, this step of the algorithm is complete. Otherwise, in case the marker has 4 detected edges, we desire to use the information of all 4 edges.

Therefore, the algorithm is applied for each possible input combination and each partial result is considered. There are 4 possible ways to combine the 4 input edges in groups of 3, and for each one of them the pose is computed using the algorithm covered in the previous section.

Adapting the Result

The camera pose was computed up to this point only taking into consideration input related to the current frame obtained from the camera. However, this algorithm of obtaining the pose is not applied once on a single image, but on the input stream of images coming from the camera, so-called frames. This means the result obtained on a local context by analyzing the current frame needs to be integrated in the global context, such that the new pose makes sense in the light of the earlier ones. In order to do that, two things need to be taken care of:

1. Adjust quadrant to previous pose;
2. Smooth out noise.

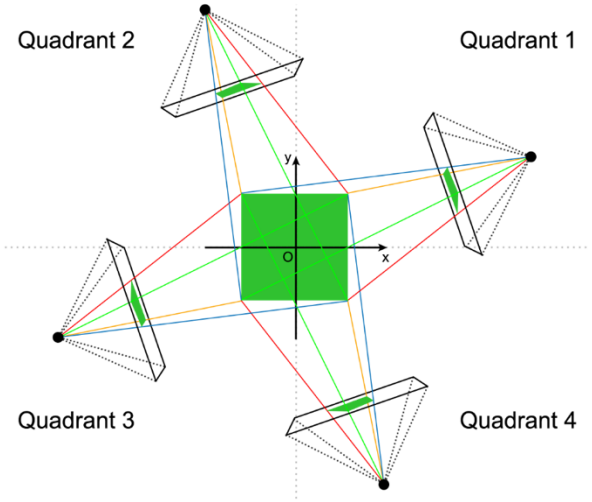

Figure 6. Pose ambiguity on a symmetrical marker; camera could be located in each of the 4 quadrants and still fulfill the marker projection conditions

Adjust Quadrant to Previous Pose

As we chose a scene marker type that would require the user minimal effort to procure it, the scene marker is merely a square piece of paper of some specific color. Being a square, we can identify its edges and compute the camera pose from this information as described earlier, however there is a

limitation: because this square is perfectly symmetrical both along the x axis as well as the y axis, there are no unique features that we could identify on it and track in order to determine the camera's absolute pose with respect to the marker.

Figure 6 illustrates how the marker's symmetry allows each pose to be accurately determined inside a quadrant but leaves the question open of which of the 4 quadrants it is actually in. Additionally, there is no way to solve this problem by just analyzing the marker edges, as the input for pose computation are the results coming from the Ransac algorithm that has a randomness factor, which leads to the order in which marker edges are considered being possibly different from one frame to the other.

To solve this problem, the solution is to ensure coherency exists between each pair of successive frames (i.e. minimal difference in orientation), without caring about our absolute position.

The only situation in which this method fails is when the difference in orientation between the pose of two successive frames is more than 45 degrees. This is a theoretical possible but practically unattainable case, considering that the application is designed to work at a minimum of 24 frames per second which would require the user to rotate the camera more than 45 degrees in less than 42 milliseconds in order to get this an error – which is extremely unlikely.

Smooth Out Noise

In order to increase the continuity between frames despite the presence of noise, each pose result is averaged with the result obtained in the past few frames. Thus, a smooth movement is obtained at the expense of accuracy. In other words, the averaged pose is more stable and steady and less influenced by noise due to the fact that it represents an estimation based on values computed from several frames, but it is less responsive to changes.

4.2 Pencil Position

Besides the marker representing the scene base (scene marker) which also represents the main system of reference according to which transforms are defined in the world, the system also scans for the pencil marker. The pencil marker is the marker representing the user's input in the 3D scene, in the form of a position in 3D space. The user utilizes the pencil to model the virtual 3D objects.

Pencil Detection

The pencil marker consists of a disk of some predefined color. It is a circle cut in colored paper that is attached to a small stick that will allow the user to move it freely through the scene without having his hands interfere with the pencil or scene markers (Figure 7). The pencil position passed as input to the program will be the center of the disk.

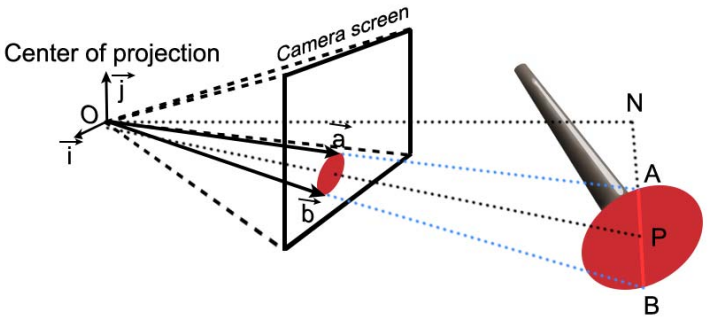

Figure 7. Pencil marker detection process

Pencil Position

At this point we have all the information needed to compute the pencil's position. The next step is first to determine the pencil's 3D space position with respect to the camera, and then, knowing the global position of the camera, compute the global position of the pencil.

Gravity

Up to this point we obtained the pencil position only by analyzing the current camera frame. However, if the user wants to interact with previously created shapes in the virtual scene, the raw pencil position input may not deliver the desired accuracy. Due to this fact, the pencil position determination process undergoes one last phase, the exposure to gravity.

By this interactive technique, the pencil position will be snapped to that gravity point, such that the user will not need to move the pencil exactly to the desired position. Therefore, the system anticipates the position where the user wants to move the pencil and, if the pencil is in its proximity, it automatically moves the pencil to the point of interest.

4.3 User Interface Input

Apart from the pencil position which defines where to draw the models in the augmented reality scene, additional input is require to define what and how to draw. This input is taken from the user interface and defines the structure, color, and parameters of the shapes to be created with the pencil.

The construction concept is to build the model out of a set of individual components. These components will be defined using a series of shape primitives which can be instantiated with custom set attributes. Regarding the shape primitives, the user will be able to choose from a menu containing basic shapes such as sphere, line with square diameter, line with circle diameter, Bézier curve, free curve, triangle, etc.

5. Output Features

Augmented Reality Scene

The augmented reality scene provides real-time feedback to the user for his actions on the UI and with the pencil marker. It offers the user a quick and intuitive way of visualizing and interacting with the model.

Nevertheless, the difference between real and virtual objects will still be visible, mainly because of different illumination. For no matter what illumination model is used to render the virtual scene, the result will still look different from the real world because:

- There is no ray tracing process. The rendering of the scene needs to be done in real-time, making ray tracing impossible to use due to its high computational cost.
- The lighting parameters of the environment are unknown. Parameters such as light sources intensities, light sources positions, ambient light and surrounding material properties all contribute to the way light falls onto an object. If these parameters are not known, then the virtual object will look unnaturally different from the real world objects and the human eye will percept it as an intruder.

Phong Reflection Model

The model represents a 3D structure in a virtual scene. In order to be visible to the user it is brought to the screen by a process called rendering, which is composed of a series of operations known as graphics pipeline. Aiming to reproduce the model as it would appear in reality, with colors, overcast

shadows, etc. we hope “to display an image that approximates the real object closely enough to provide a certain degree of realism” (Phong, 1975).

Advanced Shapes

In order to model the desired object, the user has at its disposal basic primitive shapes which he is able to instantiate with different parameters such as position, size, and color. The more challenging primitive shapes are the Bézier Curves. A Bézier curve is defined by a set of points known as control points. Due to the nature of its parametric function, it has the property that, once the control points are defined, it represents a smooth curve that runs through the first and the last of the control points and presents an approximation of the other control points in between, offering an intuitive way to describe continuous curves and a pleasant-looking result.

6. Experimental Validation

6.1 System Configuration

The application is built using the Android software development kit (Android SDK) on a Windows 10 machine. The IDE used for this was Android Studio, which is the official IDE from Google for native Android application development since 2015. Android Studio is based on JetBrains' IntelliJ IDEA software but designed specifically for Android development.

The image processing part in the application is done using the OpenCV Java library for Android. OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. The library is cross-platform and free for use under the open-source BSD license.

In order to communicate with the phone’s GPU for rendering the virtual scene, OpengGL ES 2.0 is used. OpenGL ES (OpenGL for Embedded Systems) is a subset of the OpenGL computer graphics rendering API for rendering 2D and 3D computer graphics, such as those used by video games, typically hardware-accelerated using a graphics processing unit (GPU). It is designed for embedded systems like smartphones, computer tablets, video game consoles and PDAs.

6.2 Visual Results

The user may edit 3D virtual objects working directly within the 3D real space (Figure 8). The feeling and the manipulation of the tree dimensions (i.e. x, y and z; width, length and deep) is more natural than within the known 3D graphical editors.

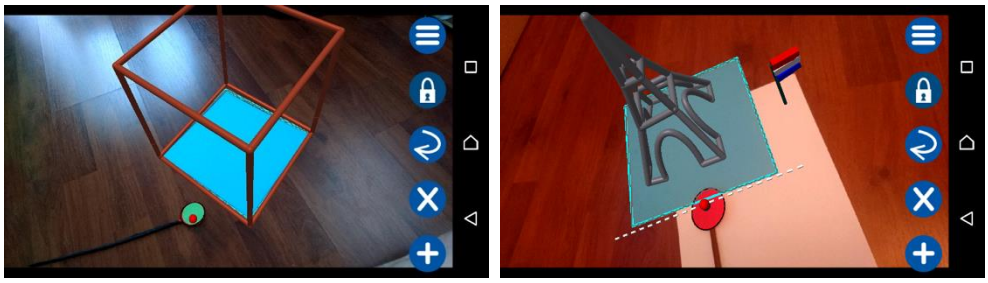

Figure 8. Building up complex 3D virtual objects by working directly within the 3D real world

6.3 Performance Evaluation

As have been already experimentally checked, the load on the GPU and main memory consumption is moderate and is not detrimental to the system's performance. The system's most important aspect is how to make use of the CPU in order to ensure real-time interaction.

The main and only relevant way to measure time performance in this application is by counting how fast the system is able to process incoming camera frames – this means the time between the frame arrival from the camera as input to the system, and the displaying of the AR scene to the user, in concordance to the frame, the model and user input. This processing is done continuously, such that as soon as a frame is finished processing and displayed the system starts working on the new one. This is the system's most time-critical path as it represents the work done for the AR scene which is the application's main feature. All the other tasks such as user input coming from the UI are short and direct, rather determining how the main task is executed and not actually doing anything.

During the application's most heavily loaded use case (the AR editor),

measurements show that the application works at an average of 17 frames per second (fps). While this value is enough to enable a steady use of the application, it is not a very good value considering that most video formats operate at a minimum frame rate of 24 fps. The frame rate does not directly affect the user experience in the AR editor as it does not lag and is still perceived by the brain as motion, however it is not perfectly fluid and does put stress on the eye indirectly and especially over time.

More than 90% of the frame processing time is spent in performing image processing on the camera frame in order to compute the camera pose and the pencil position, and, despite being heavily optimized, the resulting frame rate shows that this is a part of the application that still needs improving. As such, at this point in time the quality requirement of a minimum frame rate of 24 fps is not fulfilled.

6.4 Functional Evaluation

Static Factors

The user experience is affected by certain aspects of the environment where the application is used. The factors covered here are static – once established, they do not change during the usage of the application

Marker Uniqueness

By marker uniqueness we refer to the condition that the color of the scene and pencil marker object is not found in the environment in significant amounts.

This comes with the drawback that there are times where the color of the marker is similar to the color of the environment visible in the background, leading to false pencil and scene marker identifications. The step where the user selects the colors of the scene and pencil marker, as well as the radius in the HSV color model around which they are to be searched for, is to be treated with great care.

Illumination

Environment illumination is also an important factor since the detection of the markers is color-based. The environment is to be brightly and uniformly lit. Otherwise, the pencil marker's shadow may fall hard enough on the underlying scene marker, thus affecting its proper detection.

Marker Size

The size of the pencil and scene marker directly affects the accuracy of the camera pose and pencil position retrieval.

While the size of the scene marker should be small enough to entirely fit in the camera frame but large enough to deliver a proper estimation of the camera pose, experiments showed that this value is not an issue, as it can vary a lot (from 50 mm to 300 mm) without affecting the accuracy of the camera pose.

As for the pencil marker, it represents a colored disk of some diameter, whose value represents a trade-off. The smaller the value, the smaller the pencil marker will be. This will make the user pencil appear more natural to the user as it resembles more and more a normal pencil, and it will decrease the probability of obstructing the scene marker during the drawing phase. However, we rely on the size of the diameter to retrieve the distance of the pencil marker to the camera; the smaller this value is, the less precise our estimation will be. A value of 30 mm turned out to be a good compromise between these two options

Dynamic Factors

Besides the static ones, there are two factors influencing the correct functioning of the application that change during the usage.

Distance

The distance to the markers is closely coupled to their size. The camera must not be too close, so that the markers fits inside the camera screen. The pencil marker must always be fully visible in order to be correctly analyzed, and the scene marker must have at least 3 out of 4 edges detected, while 4 edges are desirable for an increased pose accuracy.

The maximum distance depends again on the marker's size. In the case of a pencil marker of 30 mm diameter and a scene marker with the side of 200 mm, the maximum distance is 40 mm for the pencil detection and 1300 mm for the scene marker detection. The detection of the scene marker without the pencil marker would allow the user to visualize the AR scene but not interact with it, as the pencil position is missing.

Speed

A correct marker detection is also influenced by the speed with which the camera moves through the environment, as well as the speed of the markers. However, this value turned out to be of no concern during normal use of the

application.

6.5 Usability Evaluation

The application was given to three persons from different categories to test its usability.

The first individual is a computer science student with high expertise in the image processing domain. The second person has high knowledge skills in engineering such as control theory and thus strong logical thinking is shaped, however he is not aware of the exact implementation details. The third person has little to no knowledge in the field, though she is very familiar to the smartphone technology (hence she is representative for the majority of potential users out there).

The testing process was done for each person individually. He was notified that the test was done to evaluate the usability of a research project, the purpose of the application was clarified, the way to operate it, and to some of the testers a short life demo was shown. Next, each tester was allowed to freely use the application as desired, and at the end he was given the task to draw a smiley face. First, the testers' feedback is presented and then my personal notes and conclusions on the tests.

Critical feedback from the testers

The critics and the recommendations of the testers could be concluded by the followings:

- Dependency on color in order to identify the markers is a problem; the constant care for a proper marker detection (in order not to be confused with the environment or covered by shadow) is tedious;
- More primitive shapes should be available;
- Working with the application provides a good feel for grasping the 3rd dimension;
- Additional to the color picker, a color palette should be visible where the user can quickly pick a color among a set of basic predefined colors; this would speed up the process of selecting the color in cases where the user is not interested in a very specific color;
- More customization options should be offered;

- A useful addition would be an extra functioning mode in which the angle from which the virtual scene visualized is not set according to the AR scene, but can rather be set manually by the user, using swiping gestures over the touchscreen.

Usability evaluation given by the developers

The developers performed themselves usability evaluation sessions. The conclusions and recommendations are the followings:

- The tester's prior knowledge in computer science turned out to play an important role; testers with domain knowledge understood from start what the marker detection process consisted of, and hence knew how to set the parameters accordingly. Whereas the person with highest experience figured out on her own to hide an object in the background that had approximatively the same color as the scene marker in order not to interfere with the detection process, the person with no knowledge did not really grasp the conditions necessary for a good marker detection until the end; additional effort needs to be invested in order to avoid this, by managing more of the tasks that are currently done by the user, in order for the application to be less dependent on his actions;
- A visual tutorial (at best a video) is very important to be shown before usage; it was much easier to understand what needs to be done at each step in cases in which a live demo was shown to the testers at start. This is all the more important as the application is not a standard one and greatly relies on user cooperation;
- The application proved to be very appealing and fun to the test person that had painting as a hobby;
- To ease up usability, at various points in the application the main commands are launched not by the press of one particular button, but by tapping anywhere on the screen; it is more convenient to the user to simply touch his finger anywhere on the screen than needing to hit exactly one button; however, this created confusion among all users, due to the fact that the tapping needed to be done over the AR scene, which gave them the impression that the actual position of the point where the tapping was done was important, confusing it with another form of input; additional emphasis needs to be put on the instruction

that the tapping may be done anywhere on the screen and the exact location plays no role;

- Users did not really comprehend the principle of the Bézier curve; when the control points needed to be added at each new editing step, they expected the curve to run through the control points and not in-between as it was the case; some visual representation of the control points and their relation to the curve needs to be shown during the drawing process in order to avoid this confusion in the future.

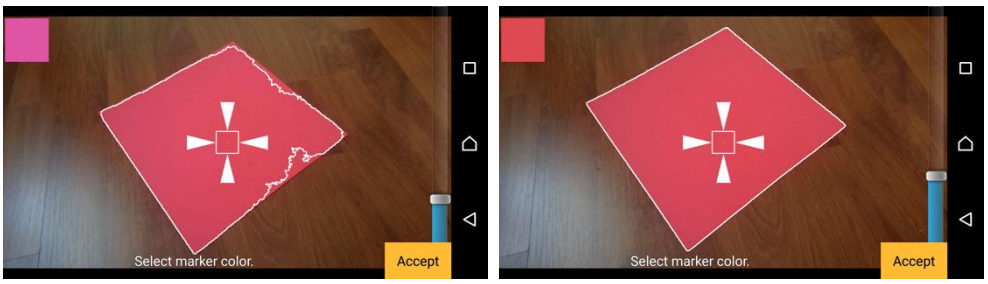

Figure 9. Marker selection: (left) search radius too small ; (right) correct color and radius parameters

7. Working Session Calibration

Using this application supposes executing 5 major steps:

1. Select the color and intensity of the scene marker
2. Select the color and intensity of the pencil marker
3. Calibrate the pencil marker with respect to the scene marker
4. Select the attributes of the shape to be drawn
5. Draw the shape in the editor

Steps 1 to 3 are done only once, at application start. Steps 4 and 5 are done repeatedly during the object modeling phase in the AR scene.

7.1 Define Scene Marker

The first step after launching the application is to select the attributes to search for when looking for the scene marker. These consist of the color to be searched for as well as the radius in the HSV color space in which two colors will be considered equal. As Figure 9 points out, the parameters shall

lead to a full marker detection but without having it interact with the environment.

7.2 Define Pencil Marker

Similar to the scene marker the pencil marker shall be defined (Figure 10.left).

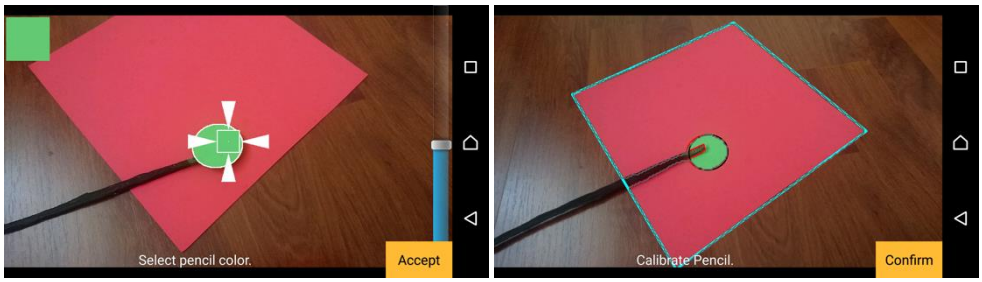

Figure 10. (left) Pencil marker attributes are being selected; (right) Calibration of marker size by placing both markers in the same plane

7.3 Calibrate Markers

Once the two markers can be tracked, one thing remains: the system must know the size of each marker relative to the other one, so that it will be able to compute their distances to the camera on the same scale, and thus their transforms in 3D space in the same coordinate system (Figure 10.right). This step is important as the camera has no depth information.

7.4 Select Shape Attributes

At this stage the main scene featuring the AR editor is visible (Figure 12.left). The detection of the scene marker and the pencil marker are done using the attributes defined previously: shape type, size, and color (Figure 12.right).

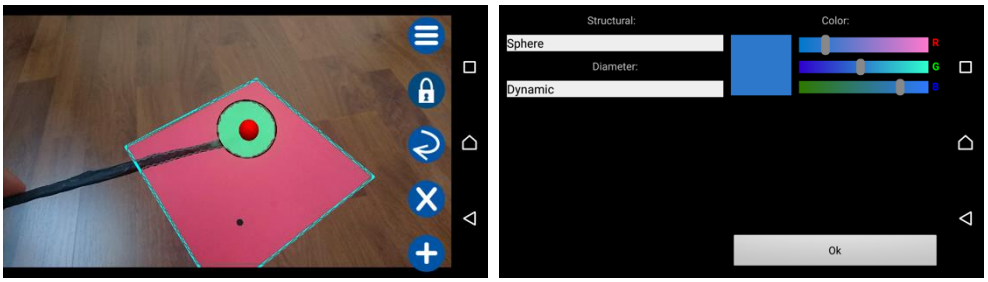

Figure 12. (left) AR editor on top of the scene marker with the pencil marker position tracked in 3D space ; (right) Shape select menu. The created shape will be a blue sphere with the diameter specified dynamically with the pencil, after its creation

7.5 Draw Shapes

Once the shape attributes are defined, the shape may be drawn on the scene (Figure 11). Tap anywhere on the scene once in order start drawing. Tap again each time to get to the next editing step. All shapes are drawn in editing steps in which the pencil position is submitted, the transition from one editing step to the other being the tap on the screen gesture. Simple shapes, as for example the drawing of a fixed size sphere, require only one editing step (specifying its position with the pencil). More complex shapes may require more editing steps, such as a square line of dynamic diameter which requires 3 (specifying the start point, end point, and the width of its section). There are shapes such as the Bézier curve that require an indefinite

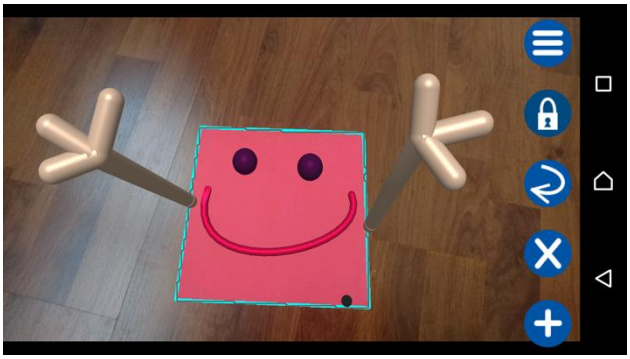

Figure 11. Drawing made with the Augmented Space Editor

number of editing steps (for the Bézier curve, with each editing step a new control point is added). In this case, a red button in the lower right part of the screen will appear which may be pressed at the end to finish editing the current shape.

8. Conclusions

8.1 Results and Issues

A new concept of human-computer interaction having as purpose the modeling of 3D objects has been designed, implemented and evaluated. The root idea is to allow the user to define and interact with multiple 3D shapes in an augmented reality context, using as input the touch screen and two markers, the first representing the scene base on top of which shapes will be drawn, and the second representing a cursor which is able to move in 3D (unlike the typical mouse cursor which can only be moved in two dimensions).

The main advantage was the ability of the system to mix the real perspective from which the scene is looked at with the virtual one, blending the two worlds together. The user was able to model shapes directly and straightforwardly, without worrying about domain-related terms that would appear in a normal graphics modeling software, such as virtual camera, coordinate system or scene depth – which where all now linked to the reality world.

There were also some issues identified while testing the system. These issues are all related to implementation details of the application, and not to the concepts themselves.

Issues preventing a problem-free usage were mostly caused by its physical implementation of the marker detection. The advantage of a color-based marker detection also turned out to be one of the system major problems. The system sometimes failed to perfectly detect markers by either not identifying them fully, or confusing them with parts of the environment background. This lead to bad pose computations which made the virtual scene look misplaced in the AR scene.

Another considerable drawback is the fact that the system relies heavily on the user to do his job right: the user needs to come up with a perfect disk to represent the pencil marker; a perfect square for the scene marker; setting

the color parameters of the markers needs to be done with optimal settings to enable a good identification later on; during the calibration phase, the pencil marker needs to be placed on top of the scene marker. These are all actions in which the system expects the user to do his task right, with limited or no ways to verify if they were done right.

As general characteristics of the system, we developed an application that favors speed and ease of use over precision. Accuracy was limited to the precision with which the scene marker and camera marker where detected, which were subject to noise. It relies on care and attention on the side of the user to use it properly in order to make it work as intended.

8.2 Further Development

The application is a proof of concept and is far from being a complete tool for modeling 3D objects ready to be released on the market. Additions may be implemented in almost every direction pointing out the potential of the underlying concept, and only a fraction are presented here:

- *Auxiliary input methods:* Besides changing the marker detection approach, the application could benefit from auxiliary ways to determine the camera's position in 3D space. This is due to the fact that sometimes it happens that the pencil marker obstructs the scene marker in such a way that a correct pose computation is not possible.
- *Space partitioning:* At the current stage, the virtual scene used to hold the data of the model solely consists of a collection of 3D shapes. In order to perform operations on the shapes, such as to compute the gravity result of the pencil's current position, we need to iterate over the whole collection and check each individual shape, which leads to an execution time of O(n), where n is the number of shapes in the scene. While this approach is genuine for a scene of medium size (runtime was never a problem for scenes containing dozens of shapes), it does not scale well for shapes of larger size (keep in mind that the computation of the pencil gravity result needs to be done at every frame, thus its execution time is critical).
- *New editing operations:* At the moment the user can create a custom model having at his disposal a series of basic shapes (sphere, line, etc.) which may be instantiated with any parameters. A custom 3D model may then be constructed by combining these shapes, as one

would build a house using several types of bricks. However, there are a lot more editing operations to be implemented using the underlying concepts. An editing operation that would offer way more flexibility to the user is sculpting. The user would then chisel the desired 3D shape in the cube, exactly as a sculptor would do on a real block of stone. A merging algorithm between two 3D objects must be employed, such the one presented in (Decaudin, 1996).

- *Save and export:* At this point, the structure modeled by the user resides only in the main memory and is lost when the application is closed. A useful expansion is the possibility to save one's work in the phone's storage, such that it can be opened later for visualization and further editing.

References

Alvarez, H., and Borro, D., 2009, A Novel Approach to achieve Robustness against Marker Occlusion, *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP9)*, pp. 478-483. Lisbona, Portugal.

Buchholz, R., 2014, Augmented Reality: New Opportunities for Marketing and Sales, KEY VALUES GmbH, Hamburg, Germany.

Dachsbacher, C., 2015, *Computergrafik course 9*, Karlsruhe Intitute of Technology, Karlsruhe, Germany, slides 29-33.

Decaudin, P., 1996, Geometric Deformation by Merging a 3D Object with a Simple Shape, *Graphics Interface '96 proceedings*, May 21-24, Toronto, Canada. [8] Fitzgibbon, A.W., Fisher, R.B., 1995, A Buyer's Guide to Conic Fitting, *Proc.5th British Machine Vision Conference*, Birmingham, UK, pp. 513-522.

Gallier, J., 1999, Curves and Surfaces in Geometric Modeling: Theory and Algorithms, *Morgan Kaufmann Publishers Inc*. San Francisco, USA, pp. 81-86.

Huot, H., Dumas, C., and Hégron, G., 2003, Toward Creative 3D Modeling: an Architects' Sketches Study, *IFIP TC13 International Conference on Human-Computer Interaction*, Zurich, Switzerland.

Lourakis, M.I.A., 2005, A brief description of the Levenberg-Marquardt algorithm implemented by levmar, Technical Report, Institute of Computer Science, Foundation for Research and Technology - Hellas.

Mackay, W.E., 1998, Augmented reality: linking real and virtual worlds: a new paradigm for interacting with computers, *AVI '98 Proceedings of the working conference on Advanced visual interfaces*, pp. 13-21.

Phong, B. T., 1975, Illumination for computer generated pictures, *Communications of ACM 18 no. 6*.

Schmalstieg, D., 2016, Hollerer, T., Augmented Reality: Principles and Practice, *Addison-*

Wesley Professional, pp. 28-29.

Siltanen, S., 2012, Theory and Applications of Marker-based Augmented Reality, *VTT science*, vol. 3, pp. 51-53.

Szeliski, R., 2010, Computer Vision: Algorithms and Applications, *Springer-Verlag New York*, Inc. New York, USA, pp. 91-92.