Video Anamorphosis

Rok Cej

University in Ljubljana, Faculty of Computer and Information Science, Večna pot 113, 1000 Ljubljana

Abstract. This report describes the creation of a distorted image or video that looks perfect when projected onto a given surface and viewed from a predetermined angle. It utilizes the depth sensor on an Xbox One Kinect and a projector. The program is written in C++ and it starts off by recreating the projection surface in 3D. It then uses the surface model to create an anamorphic projection. If the Kinect and the projector are properly aligned, the projected image or video creates an anamorphic illusion in real life.

Keywords: Anamorphosis · Kinect · Depth sensor · Optical illusion.

1 Introduction

For the final project in my Interaction and Information Design class I chose the topic Video Anamorphosis. The goal of this project is to project an image or a video onto any surface in such a way that it looks perfect when viewed from a specific angle. In this report I will explain how I approached the problem and how my solution works.

1.1 Anamorphosis

Anamorphosis refers to a distorted projection that requires the viewer to occupy a specific vantage point in order to see a recognizable image. Earliest known examples of anamorphosis date all the way back to the Renaissance period, when artists' experimentation with optics and perspective led to development of anamorphic imagery. In the past, the process of extreme anamorphosis has been used by artists to hide certain elements of their paintings in plain sight, only revealing them to knowledgeable spectators. Nowadays, however, artists utilize anamorphosis mainly as a means of creating optical illusions. One example of such can be seen in Figure 1.

1.2 Motivation

When using a projector, the projected image looks accurate if viewed from the point of view of the projector, but if viewed from anywhere else, it looks distored. Since the projector occupies physical space, it's impossible to view the image from the exact point of view of the projector, which means projected images will

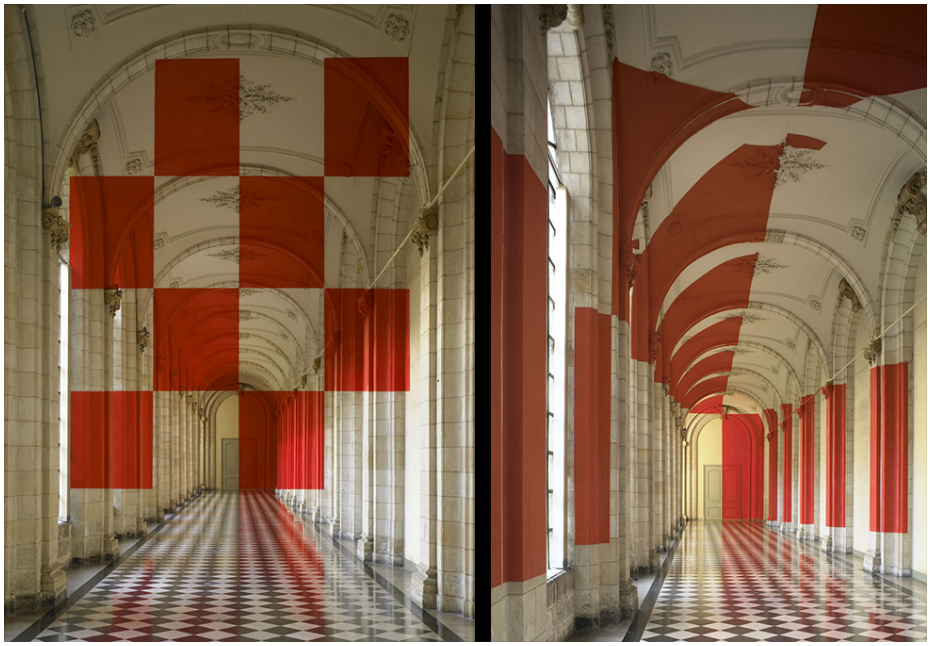

Fig. 1. An anamorphic optical illusion [1].

always be at least somewhat distorted. If the projection surface isn't perfect, the distortion is usually even greater. Can we reverse this effect? Can we intentionally distort the projected image in such a way, that it looks accurate from an arbitrary point of view, regardless of what kind of surface we are projecting it on? In other words, can we algorithmically create an anamorphic image? I thought this problem was interesting because it posed a unique mathematical challenge and included my fascination with computer graphics.

1.3 Related works

The idea for this project was suggested by my professor, Franc Solina, who also worked on a few similar projects.

His project "Dynamic Anamorphosis"[2] introduces dynamic anamorphosis, which changes anamorphic deformation in response to the movement of the observer's head. The result is an art installation where the subject of the projected image is a human face whose gaze follows the observer as they move.

Another similar project of his is "Light Fountain"[3], which uses a Kinect sensor to detect the shape of the surface below it. It then simulates water droplets and projects them back onto the surface, creating the illusion of physical water droplets interacting with it.

2 Technical requirements

2.1 Hardware

For this project I used two external devices. The first one was an Xbox One Kinect. It has a camera and a built-in infrared depth sensor that works for distances between 0.5m and 4.5m. By using Kinect I can detect the shape of the surface that I will be projecting on. It was originally designed as an accessory for gaming, but due to its usefulness it became popular in other fields as well.

The second device was a projector. After the anamorphic image is constructed, I need a projector to project the image onto the surface. The projector didn't have a specified field of view, so I had to measure it by hand.

2.2 Software

I developed the project in C++. Initially I considered using a more high-level language such as Processing or Python, but in order to avoid any potential performance issues I chose C++ instead. I used the following libraries:

- **OpenGL**: Graphics API
  - **GLFW**: OpenGL context creation
  - **GLEW**: OpenGL extension loading
  - **GLM**: Matrix and vector operations
- **Kinect SDK**: Windows Kinect API
- **FFmpeg**: Decoding video files
- **stb_image**: Reading image files

3 Creating an anamorphic image

Creating an anamorphic image isn't a trivial task. I split the main problem into several smaller ones, so I can do it step by step.

3.1 Getting depth data

After initializing the Kinect, we can start polling it to see when the next depth frame is available. Depth frames are stored in the form of a 512x424 read-only buffer of unsigned 16bit integers. Each depth value corresponds to millimeters of distance. If we map depth values to grayscale values, we can create an image that represents depth data, where brightness increases with distance. Depth values that Kinect wasn't able to detect are seen as black pixels. An example of this can be as seen in Figure 2.

3.2 Approximating missing depth data

Kinect can't always detect every single depth value. That can occur due to various factors, for example, the target being outside of Kinect's detection range, the surface being too reflective to infrared light, noise, etc. If we want to get a complete model of the surface, we need to approximate those missing values. The method I used is to traverse all possible vertical, horizontal and diagonal lines in the depth buffer, then fill the gaps by interpolating between the nearest two valid values on that line. Every missing value is approximated as the weighted sum of interpolations in four different directions. This method can produce some artifacts where big sections of data are missing, but otherwise it seems to be fairly accurate. The result of this method on data from Figure 2 can be seen in Figure 3.

Fig. 2. Kinect depth data converted to an image.

Fig. 3. Kinect depth data from Figure 2 with approximated missing data.

3.3 Converting depth data into 3D points

Once we have the depth data, we can convert each depth value into a point in 3D space with the following equation:

$$\vec{position} = depth * \begin{bmatrix} (\frac{2x}{width-1} - 1) * tan(\frac{fov_x}{2}) \\ (\frac{2y}{height-1} - 1) * tan(\frac{fov_y}{2}) \\ 1 \end{bmatrix} \tag{1}$$

where:

$$\begin{aligned} depth &= \text{depth value} \\ x, y &= \text{depth value row and column} \\ width, height &= \text{depth sensor horizontal and vertical resolution} \\ fov_x, fov_y &= \text{depth sensor horizontal and vertical field of view in radians} \end{aligned}$$

This allows us to recreate any surface in 3D by creating a virtual point cloud. Kinect has a built-in function that maps every depth value to a pixel on its camera, which means we can also obtain the colour of each point in addition to its position. I wrote a function in my program that lets me navigate around the point cloud and observe it from various angles, and the result can be seen in Figure 4.

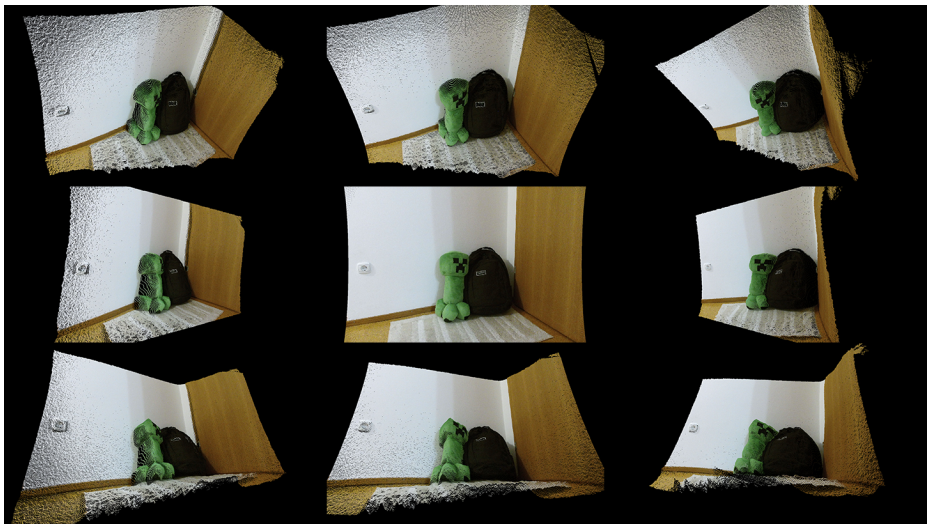

Fig. 4. Point cloud observed from various angles.

3.4 Virtual anamorphosis

Now that I had a point cloud, I decided to achieve the anamorphic effect in virtual space first, before moving on to the real world. First off, I placed a virtual observer at a vertical angle above the projector. Then, I imagined that the observer was projecting an image onto the surface - the image would seem perfect to the observer, but would look distorted from the point-of-view of the projector. For each point in the point cloud I calculated the direction from the observer to the point and determined, where that direction intersected the image. Essentially, this allowed me to map each point in the point cloud to a pixel on the image. The result was an anamorphic effect - a distorted image seemed to appear in the point cloud, however, as we moved towards the position of the virtual observer, the image would slowly become perfect. An example can be seen in Figure 5.

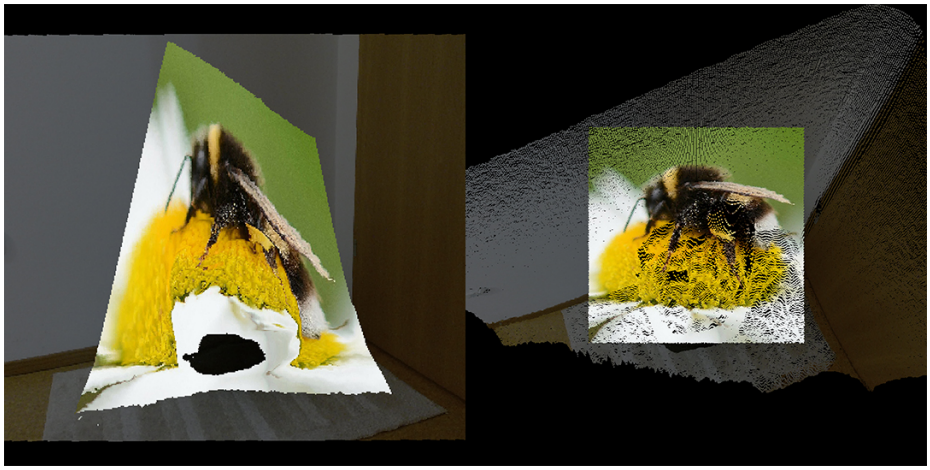

Fig. 5. Virtual anamorphosis as seen from the projection angle (left) and from the anamorphic angle (right).

3.5 Real anamorphosis

In order to achieve anamorphosis in real life, I had to compute the image that the projector would output. I started by going through every pixel of the projector and calculating which direction it faces in 3D space. The idea was to use that direction to figure out where that pixel lands on the surface, and then map that intersection point to a pixel on the image. Since the resolution of the projector was different (and much higher) than the resolution of the depth sensor, most pixels didn't align with surface points, which means I had to approximate the intersections. I did that by taking the 4 nearest surface points that the pixel lands

between and computing their average as a weighted sum. After obtaining the intersection point and taking into account the position of the virtual observer, I was able to map the projector pixel to a pixel on the image. This whole process was quite slow, so I optimized it using multithreading - since projector pixels are independent of each other, they can be processed in parallel. An example of a projector output can be seen in Figure 6.

Fig. 6. Anamorphosis in the point cloud (left) and the anamorphic projector output (right).

4 Calibration

Before combining the Kinect and the projector to achieve an anamorphic effect in real life, they have to be calibrated. I implemented a function that draws a red square in the point cloud, which represents where the Kinect expects the projected screen to be. The user can then manually adjust the position of the Kinect or the projector, so that the red square aligns with the projected screen. When they are aligned, it means the devices are calibrated.

5 Results

My program supports both video and image anamorphosis. After all, they aren't that much different - video anamorphosis is basically just image anamorphosis where the image changes very rapidly. Figures 7 and 8 show examples of image anamorphosis that I created at home and recorded with my phone. For examples of video anamorphosis that I made, you can watch this video on YouTube: `https://youtu.be/_eypZlZTRcM?t=148` [4].

Fig. 7. Anamorphosis on a board: projector view (left) and observer view (right).

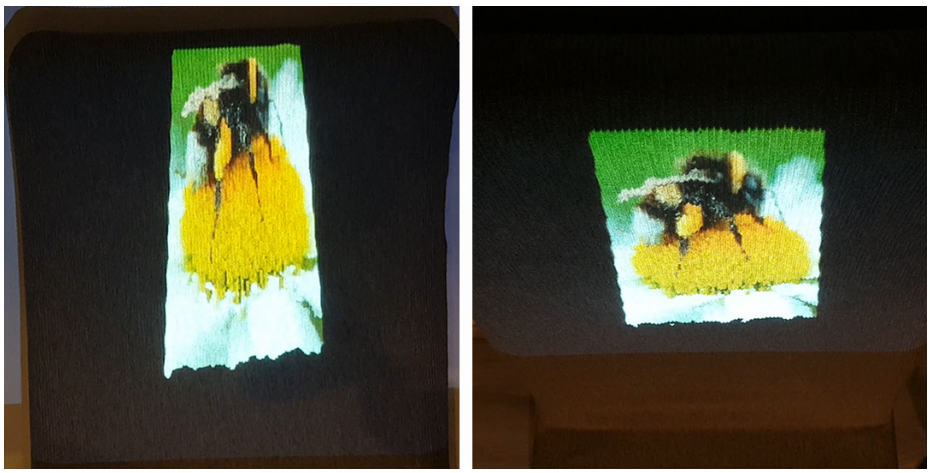

Fig. 8. Anamorphosis on a chair: projector view (left) and observer view (right).

6 Conclusion

I am satisfied with the results, but there are still some imperfections. It seems there are slight errors present with Kinect depth readings, which are big enough to make visible inconsistencies in projected images. Nevertheless, I feel like I learned a lot by working on this project, especially on how to use Kinect, since it was my first time using it. This project also has the potential to be used as an artistic exhibition that creates an interactive anamorphic optical illusion in real-time.

References

1. Felice Varini. 2007. Huit rectangles. http://varini.org/varini/02indc/30indcb07.html. Last accessed 17 Jan 2020.
2. Franc Solina and Borut Batagelj. 2007. Dynamic anamorphosis.http://eprints.fri.uni-lj.si/996/. Last accessed 17 Jan 2020.
3. Franc Solina and Blaž Meden. 2016. Light fountain – a virtually enhanced stone sculpture, Digital Creativity. https://doi.org/10.1080/14626268.2016.1258422.
4. Video Anamorphosis, https://youtu.be/_eypZlZTRcM. Last accessed 18 Jan 2020.
5. Anamorphosis, https://en.wikipedia.org/wiki/Anamorphosis. Last accessed 18 Jan 2020.
6. Kinect SDK Tutorials, https://homes.cs.washington.edu/~edzhang/tutorials/index.html. Last accessed 18 Jan 2020.
7. Video App, https://github.com/bartjoyce/video-app. Last accessed 18 Jan 2020.