Videocubes: Video as Geometry

Arthur Brussee

UCL, COMP0028_18-19

---

1. Introduction to videocubes

Timelapse video is a well studies topic in Computational Photography [1] [2]. In lectures for the course COMP0028_18-19 taught by Tim Weyrich a key insight for timelapse videos was taught [3]: videos can be viewed as a 3 dimensional entity. It is an object with two spatial dimensions, and one temporal dimensions. Just like normal image-processing works in the spatial dimension, we can apply similar operations in the temporal dimension to allow for effects such as a virtual shutter speed, or non-uniform timelapses.

This concept can be taken quite literally: a video is a 3 dimensional cube. More specifically, We can view every pixel as a little voxel - a little cube of 6 vertices with some spatial and temporal extent. Visualizing a video with all these voxels however would only show the outer voxels. Figure 1 shows an example of a video taken at the Shibuya crossing in Tokyo (video credit: Shibuya Crossing Full HD, by Anubi1989, Published on Oct 10, 2011). The geometry is not very interesting to look at and it's hard to gleam any information about the video from this.

Figure 1: A video with every voxel in the video visualized. Since only the outer layer of the video is visible, not much information can be gleamed from this videocube.

We could improve this, and try to extract geometry for only particular aspects of a video. Let $f_{rgb}(x, y, t)$ define a color for each pixel in the video. Now define some mapping $M : \mathbb{R}^3 \to \mathbb{R}^4$ such that we obtain a new function $g_{rgba}(x, y, t)$. This g function can be plugged into the visualizer instead. The added channel acts as transparency function so we can peak inside the video cube. This report will show an implementation of this concept, and explore different mappings to see what we can gleam from them.

2. Relation to computational photography

The go to solution to visualize video processing algorithms is to display a few frames in a report, or maybe link to small pieces of supplementary video material. When doing a report on old film restoration I realized how frustrating it is to visualize temporal sequences this way. When visualizing eg. the blotch detection algorithm I could only pick out a few frames and maybe highlight some example results, but there was no good way to show a broad overview that humans can easily intuit as 'good' or 'bad'.

When learning about the concept of the 'video cube' as used in timelapse videos, it made me wonder how much of a video could be visualized in a geometric way. Humans are very visual beings; in my work I have always found it highly advantageous to make good visualizations of the problem.

Imagine we interpret the blotch detector as mentioned earlier as some mapping and visualize it as a videocube. This would easily show lots of useful properties of all detected blotches (do they have a reasonable shape? Are they only there for a brief amount of time? Are they spaced out evenly, both spatially

and temporally?). The videocube could be a good tool to get a better intuition for many computational video problems.

It also shouldn't be understated however, that besides these practicalities another goal was to just generate some aesthetic images. My background is in the games industry as a graphics programmer, working on the intersection of technology and art. Though seemingly rarely acknowledged, art is at least to me an important driving force of research; a creative process chasing supposedly superfluous work can lead to real results.

3. Implementation details

For each pixel for each frame in the mapped video a voxel is spawned. Its color and transparency values are set to the color in the map. Transparent objects are traditionally hard to render, because their blending function is not order independent. Sorting objects correctly can be expensive or even impossible depending on the geometry. Therefore instead, voxels are simply scaled depending on their alpha value.

The videos were processed at 480x320 resolution. At 30 frames per second this means there are 4.608.000 voxels in a second. To visualize 5 seconds of film requires 23.040.000 voxels or 138.240.000 vertices. Even on high-end modern hardware that is an infeasible amount of geometry to render at interactive rates. Luckily, we can easily cull away a large amount of voxels:

1. Every voxel that is less opaque than some threshold, so any voxel where $c(x, y, t)_\alpha < c_0$. It is sometimes helpful to this threshold much higher than a small ϵ to achieve certain effects.
2. Every voxel that s fully surrounded by fully opaque voxels, so any voxel where $min_{n \in N}(g(x + \delta_n x, y + \delta_n y, t + \delta_n t)_a) == 1$

The videocube system was implemented in the Unity engine. To render the voxels a library called TC Particles was used [4]. This is a commercial library I created years ago for VFX in games. It contains functionality to render particles and program behaviours for particles on the GPU. First, all video frames are loaded into one large $x \times y \times z$ memory buffer on the GPU. For a 5 second clip these buffers are a couple hundred MBs in size. Then, a shader is run which can read from this buffer and calculates the mapped result g_{rgba}, and writes it into another 3D buffer. For iterative algorithms (eg, applying a Gaussian blur filter after the first map), we can reuse these same buffers and ping-pong between them to save GPU memory. Afterwards another shader is run that finds every voxel in g_{rgba} matching the two criteria above. The appropriate amount of voxels are then spawned, moved to the right locations, and have their colors updated. This approach is fairly brute force, but since it needs to be run only once to extract the geometry and is hardware accelerated throughout it works well; It is likely though that a quite fundamentally different approach is needed if one would want to extract geometries on very high resolution videos (eg. full HD / 4K videos).

After these steps the normal Unity / TC Particles rendering methods take care of the visualization. This means we can draw the voxels with lighting, shadows, ambient occlusion etc. at real-time frame rates.

4. Results

This section will show some images of some maps and their results. However, given their 3 dimensional nature, they are best viewed interactively. A supplementary web-page is available at https://arthurbrussee.github.io/videocube where a program can be downloaded to view/adjust videocubes interactively.

To start, consider a very simple map, which visualizes every n-th frame of the video:

$$g(x, y, t) = \delta_{t \bmod n, 0} f(x, y, t)$$

Where $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise. Figure 2 shows this geometry.

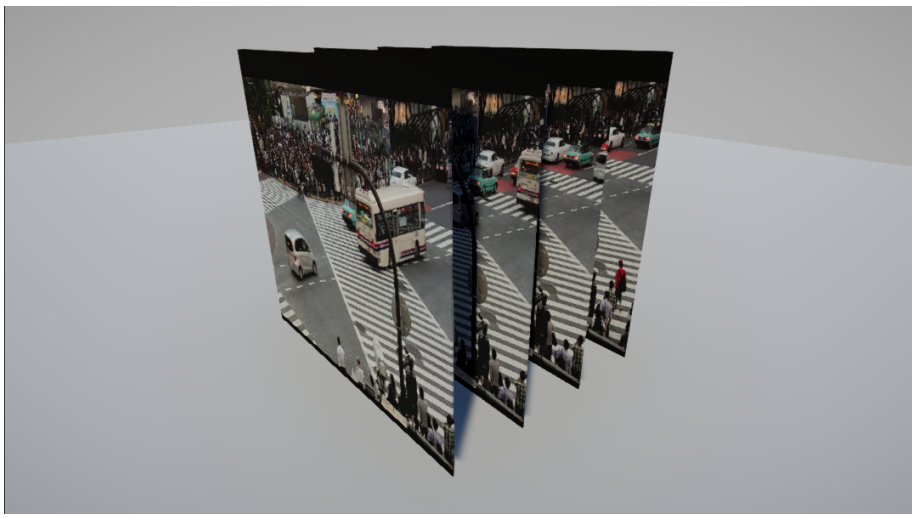

Figure 2: A simple map taking a few slices from a video

Still not very exciting. Rather than slicing in a spatial direction we can also take slices in a temporal direction, or some mix thereof. Figure 3 shows a few of these slices taken with planes at different normals.

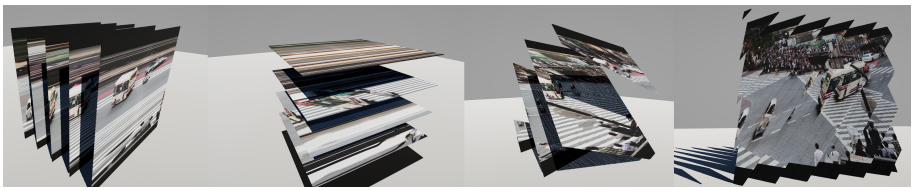

Figure 3: A mapping that takes some slices of the video at different angles

We can also visualize traditional image processing filters. Figure 4 shows a mapping where the transparency of the video is set to the magnitude of a Sobel filter.

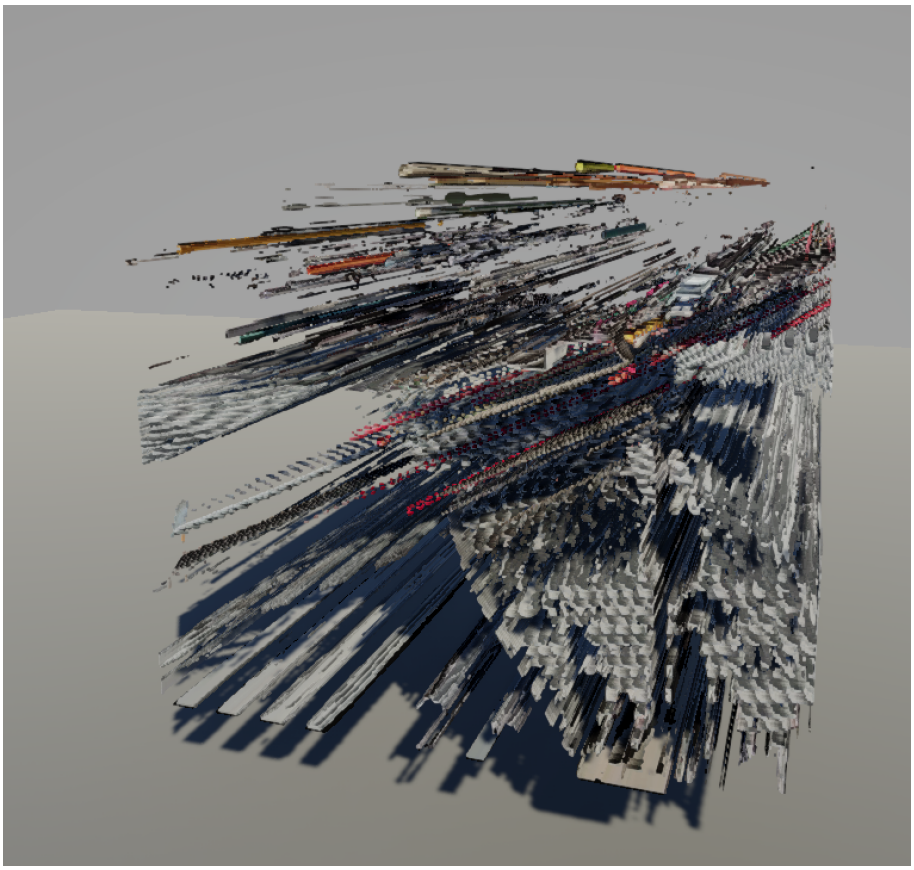

Figure 4: A mapping that for each frame calculates a Sobel filter. Since most edges don't move the appearance is a 'spiky' cube

In this mapping most edges are static but some moving objects causes streaks.Besides these 2D image filters, we can of course use filters that extend explicitly to the temporal domain. Consider the following map:

$$g(x,y,t)_{rgb} = f(x,y,t), g(x,y,t)_{\alpha} = |f(x,y,t) - f(x,y,t-1)|$$

This map roughly assigns the alpha value as the difference in pixels frame-to-frame, thus showing all pixels in the video that change over time. We'll call it the 'temporal difference' map. The visualization in figure 5 uses this maps to reveal moving objects in the scene. We can see first a bus and some cars crossing, after which humans cross leading to erratic geometry.

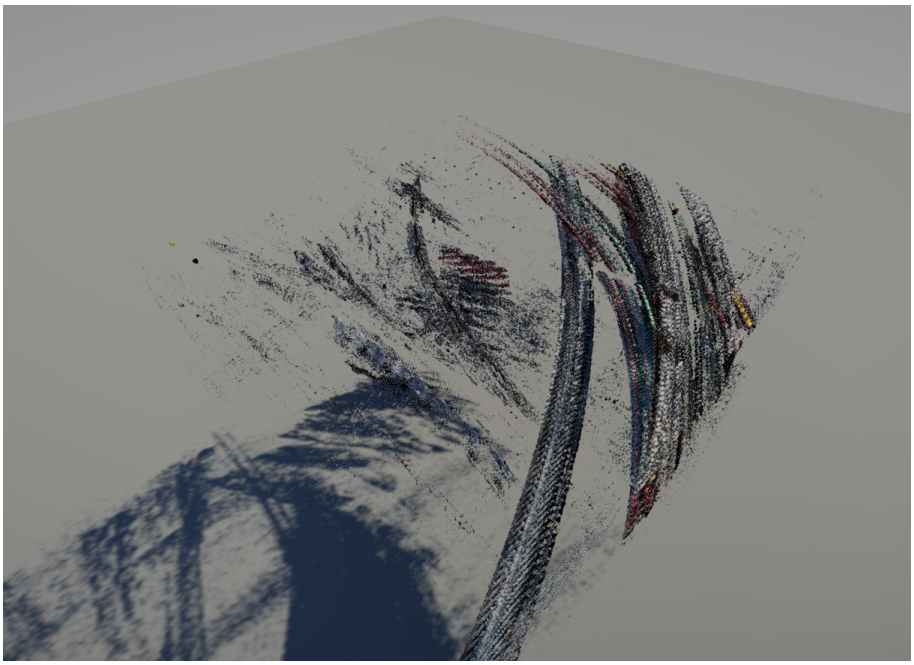

Figure 5: A mapping that shows pixels that have changed since the last frame. This reveals moving objects and their trajectories in the video

Most of these filters are quite noisy. They can be cleaned up by applying a 3D gaussian convolution (instead of a rectangle of convolution weights we now have a cube of convolution weights).

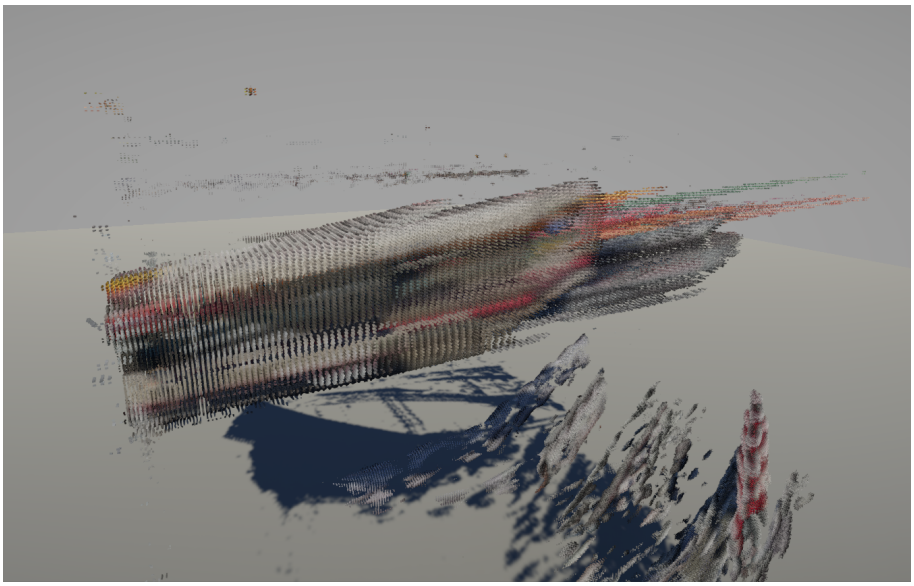

Figure 6: Same image as figure 5, but now after applying a 3D gaussian convolution to the cube. Notice the blurrier appearance of the video

This method can reveal some interesting motion in videos. Figure 7 shows this method applied to a video of a colorful newton's cradle (Video credit: 'Newton's Cradle with Pool Balls', by The King Of Random, published on Sep 18, 2018). The path of each ball over time in the cradle is nicely visible.

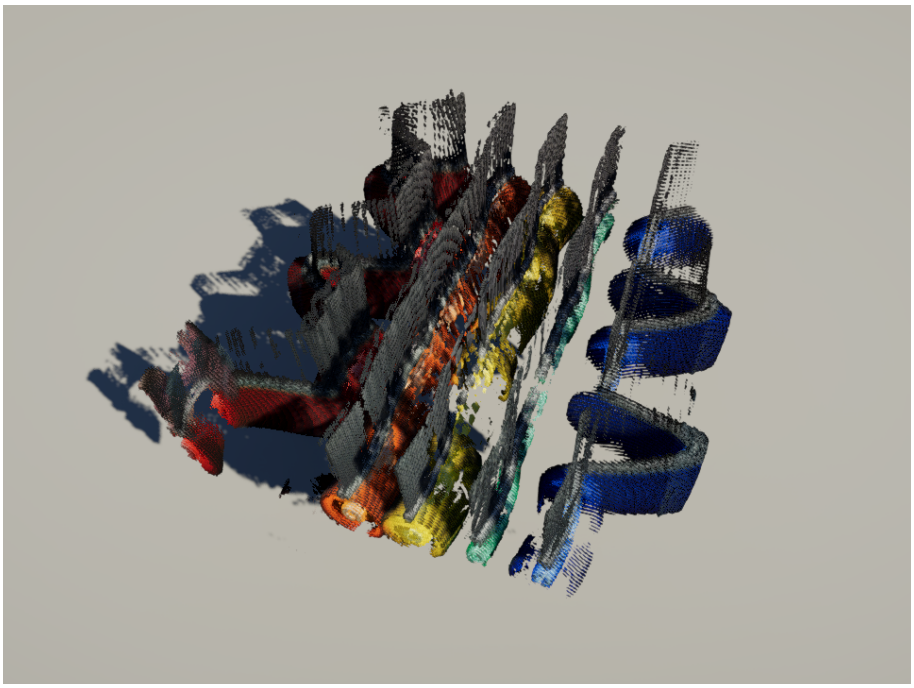

Figure 7: Video of a colorful Newton cradle using the temporal difference mapping. The trajectory of each ball can be seen.

The visualizer was also applied to a videos captured by myself. The capture was done on a Pixel 2 XL Android phone. The first mapping used is a small clip of a traffic light at night. The mapping used simply sets $g(x, y, t)_a = \text{Luminance}(f(x, y, t)_{rgb})$ and a high threshold. The result is that just the bright part of the stoplight is visible.

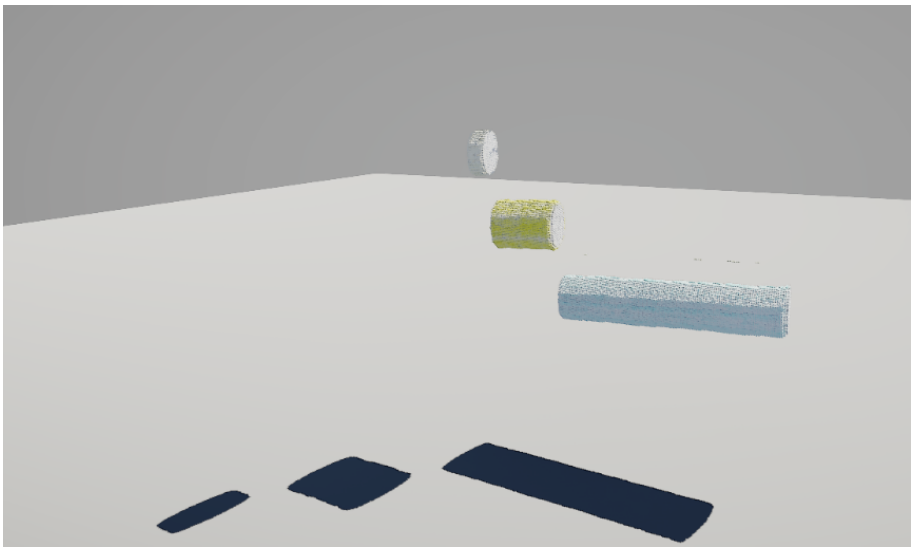

Figure 8: Traffic light going from green to red. A luminance mapping was used. The geometry shows the traffic light sequence

Figure 9 shows one half of a rotation of the London eye (with the original intent being to show one rotation, but cut to one half due to unfortunate battery issues). The mapping used is the the color difference mapping as mentioned earlier. This nicely reveals the spiralling spacetime trajectory of each of the image carts.

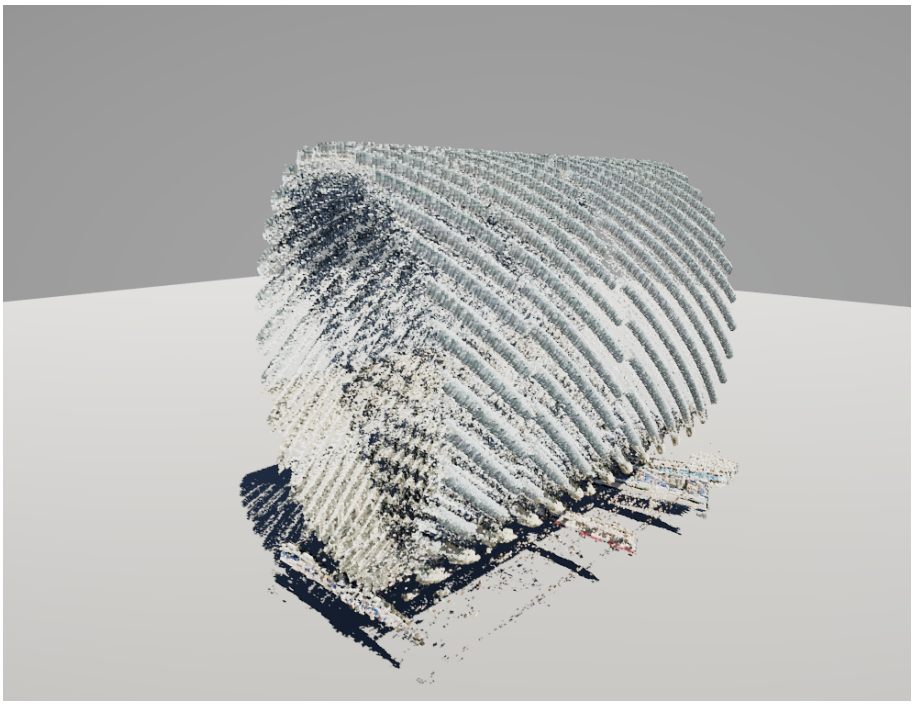

Figure 9: One half of a rotation of the London eye. Each cart traces out a spiral path over time. The noise at the bottom are boats in the Thames passing by. The gaps in the spiral are likely due to a glitch in the original timelapse video.

Besides working on videos, this method holds some value for static images and image processing. Imagine we have some normal 2D filter for a static image. One could make a 'video' where we change one of the parameters for each frame. Then using the temporal difference map, we can view the delta in the image for each change of this parameter. Figure 10 shows each frame consisting of a european flag (image credit: WikiMedia commons) blurred with an increasingly large gaussian. Since the gaussian converges to a flat blue plane, the geometry slowly fades out as each iterations doesn't change the image anymore.

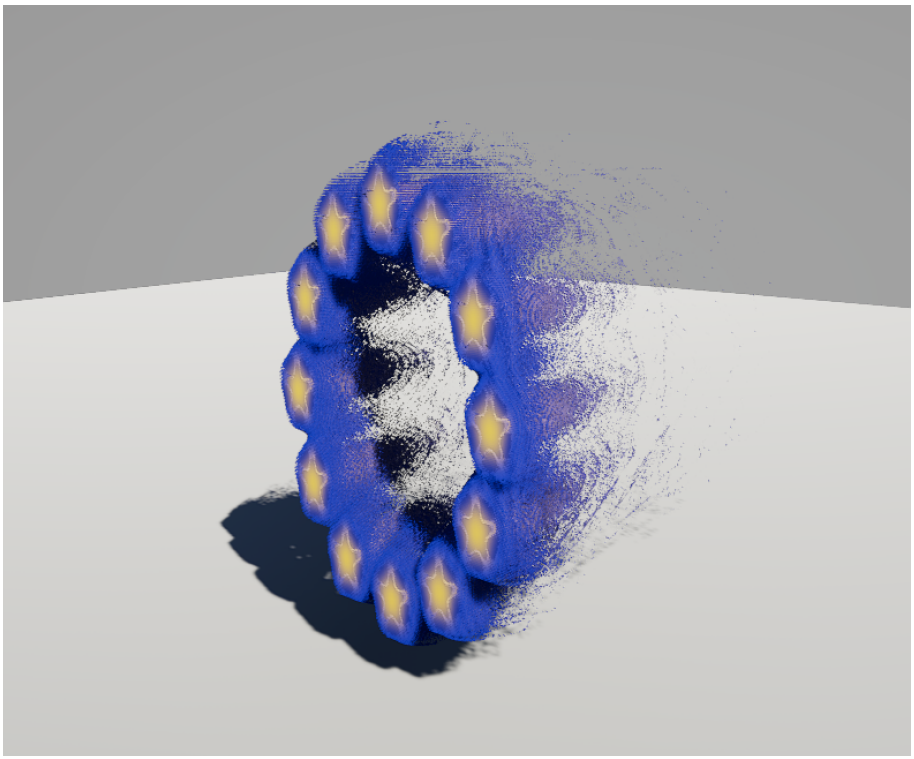

Figure 10: Result of blurring an image with an increasingly large gaussian kernel. Each slice of the cube shows the delta from the previous image.

5. Future work

As the concept of the videocube is quite generic, many other directions are still to be explored. One can image many other maps that could be implemented:

- A map taking the difference between two videos. Mostly useful to show what the results are of applying a transformation to a video.
- A map that shows each frame a jump-cut happened.
- A map that determined semantic labels and takes a subset of the semantic labels to show their trajectories over time.
- A map that sets transparency to the length of the motion vector at that pixel. This could more accurately pick out moving objects.

Several extensions could also be made to the visualizer as well:

- Expand / shrink frames. Many temporal regions might not be very interesting. The geometry could be scaled locally to make these voxels smaller.

- Interpret the geometry as a volume instead of as transparent voxels, and use a volumetric ray tracer. This would make the video act like a 'cloud' of stuff.
- Support HD / 4K resolution videos.
- Use the mapper + visualizer on the last N frames of a realtime CG rendering, to debug graphics effects.

Lastly, now that we can create interesting geometric shapes from video, there's an opportunity to manifest a videos into real life by 3D printing it's geometry. Of course, not every map $c(x, y, t)$ will result in a geometry that can be printed (eg. it might have disjoints parts), but a carefully constructed map could be.

6. Bibliography

[1] E. P. Bennett, L. McMillan, Computational time-lapse video, ACM Trans. Graph. 26 (2007).

[2] N. Joshi, W. Kienzle, M. Toelle, M. Uyttendaele, M. F. Cohen, Real-time hyperlapse creation via optimal frame selection, ACM Trans. Graph. 34 (2015) 63:1–63:9.

[3] G. J. B. Tim Weyrich, Ucl computational photography course (2019).

[4] A. Brussee, Tc particles documentation (2013).