Parallel computing on 2D&3D object morphing with AR exhibition

Author: Junlin Li (#1628363) Tianqi Li (#1628799)

Abstract

In this project, a topic of morphing is considered and it is to be implemented via parallel computing to speed up the computation process. But what makes it special is that the after-morph shape will be exhibited in an AR environment in HoloLens glasses. Then the result of morphing can be shown vividly in our real world (mixed reality).

Background

Morphing
Morphing on two function based object/shape is the idea of combining these two functions. Say there are two functions, $f_1 = 0$ and $f_2 = 0$, without loss of generality. Take a weight t (range from 0 to 1 by a weight/time step Δt), combine f_1 and f_2 linearly and obtain a result $f' = f_1 \times (1 - t) + f_2 \times t$. Therefore f' is a new function containing the information of those two functions. If we let $f_1 = 0$ and plot it onto a grid, then we can get a surface of the intermediate status in time t of the morphing process.

If f_1 and f_2 are in 2D, then f' will be in 3D. If f_1 and f_2 are in 3D, then we can only get a sequence of intermediate 3D objects which are governed by function f' with respect to the time sequence t $\in [0,1]$.

AR
AR stands for augmented reality. By the definition from Wiki, AR adds or subtracts (mask off) things in our real world to alter one's current perception of a real world environment, whereas virtual reality replaces the real world environment with a simulated one. AR can also be said as mixed reality.

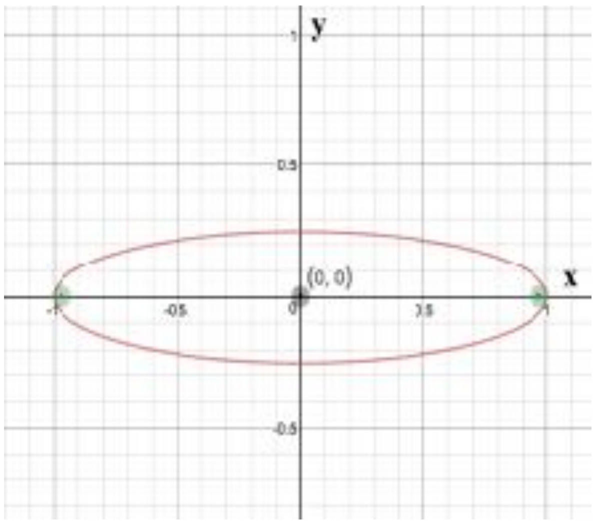
Device
HoloLens from Microsoft VR/AR device. There is a very nice app on HoloLens now and it is called Hologram. It has various type of 3D models, such as astronaut, rainbow, island, elephant, etc. And they are all well-built and very vivid. However, those models are pre-build by Microsoft, which means we cannot add new models if we have some at hand to exhibit. Then we decided to build it ourselves.

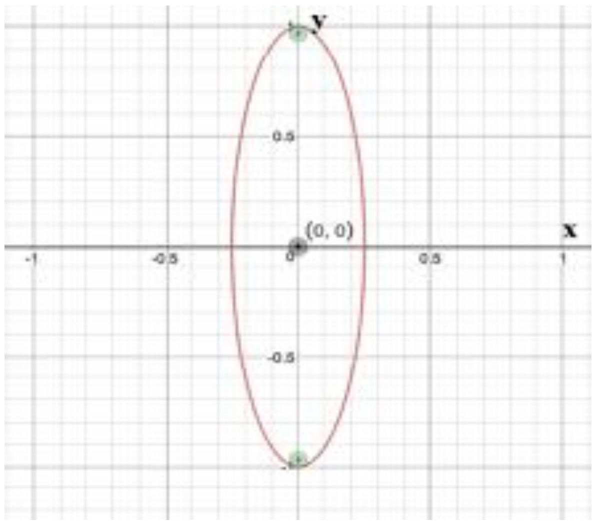
Achievement

What have been achieved in this project is implementing parallel computation using CUDA on 2D and 3D morphing and use the HoloLens device to exhibit the results.

Chosen function:
The reason of choosing such functions is that they are easy to implement and smooth. We tried weird shapes but it didn't work well. (Crazy and rough shape) Another consideration is that the point of the project is to show a huge speed up in parallel computing which means using a dense grid point and high dimension calculation are enough to observe such a huge speed-up. Last but not least, exhibiting the result in AR is the main point of doing such a project.

Functions:
For 2D, we used two ellipses, one with the semi-major axis on x-axis and the other one with the semi-major axis on y-axis.



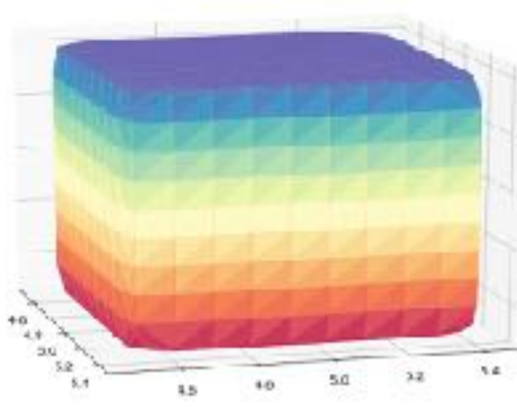


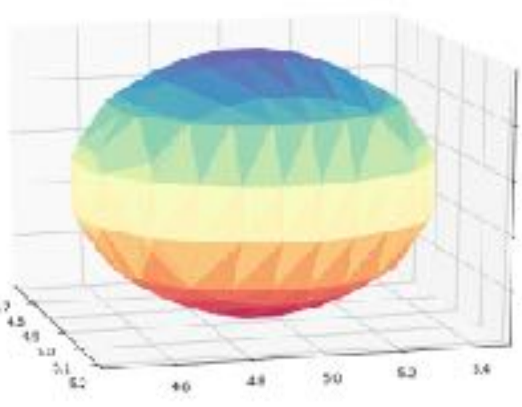
Functions are as follow:

(1) $f_1(x,y) = x^2 + \frac{y^2}{0.25} - 1 (a=1, b=0.5)$ (left picture)

(2) $f_2(x,y) = \frac{x^2}{0.25} + y^2 - 1 (a=0.5, b=1)$ (right picture)

For 3D, we used a cube with round corners and an ellipsoid (with semi-major axis on x-axis).





Functions are as follow:

(1) $f_1(x, y, z) = x^{10} + y^{10} + z^{10} - 1$

(2) $f_2(x, y, z) = \frac{z^2}{0.5^2} + \frac{x^2}{1} + \frac{y^2}{0.5^2} - 1$

Parallelism

The first part is parallelism.
In the case of 2D, serial version is just 3 nested loops (x, y, and t (weight)).
Parallel version is as follow:

```
==========================================================================
# This function concrete f_1 and f_2 in z dimension
# w ∈ [0, 1]
# f' = f_1 × (1 − t) + f_2 × t
# ellipse_x(x, y) and ellipse_y(x, y) are those two ellipse functions
@cuda.jit(device=True)
def compose_par(x, y, t):
    if t >= 0 and t <= 1:
        return (1 - t) * ellipse_x(x, y) + t * ellipse_y(x, y)
    else:
        return 100

@cuda.jit
def compose_kernel(d_x, d_y, d_t, d_f):
    i, j, k= cuda.grid(3)
    nx, ny, nt = d_f.shape
    if i < nx and j < ny and k < nt:
        d_f[i,j,k] = compose_par(d_x[i], d_y[j], d_t[k])

def parallel_compose(x, y, t):
    nx, ny, nt= x.shape[0], y.shape[0], t.shape[0]
    d_x = cuda.to_device(x)
    d_y = cuda.to_device(y)
    d_t = cuda.to_device(t)
    d_f = cuda.device_array((nx,ny,nt), dtype=np.float32)
    gridDims = ((nx + TPBX - 1) // TPBX,
                (ny + TPBY - 1) // TPBY,
                (nt + TPBZ - 1) // TPBZ)
    blockDims = (TPBX, TPBY, TPBZ)
    compose_kernel[gridDims, blockDims](d_x, d_y, d_t, d_f)
    return d_f.copy_to_host()
==========================================================================
```

In the case of 3D, serial version is just 4 nested loops (x, y, z, t (weight)).
Parallel version is as follow:

```
==========================================================================
# C is the constant that is used in the function f(x,y,z) - C =0
# parallel morph
@cuda.jit
def morph_kernel(d_f, d_x, d_y, d_z, t, C):
    i, j, k = cuda.grid(3)
    nx, ny, nz = d_f.shape
    if i < nx and j < ny and k < nz:
        d_f[i,j,k] = f_cube(d_x[i, j, k], d_y[i, j, k], d_z[i, j, k], C) * (t) + \
                    (1-t) * f_ellipsoid(d_x[i, j, k], d_y[i, j, k], d_z[i, j, k], C)

def morph3D(x, y, z, t, C):
    nx, ny, nz = x.shape
    d_x = cuda.to_device(x)
    d_y = cuda.to_device(y)
    d_z = cuda.to_device(z)
    d_f = cuda.device_array((nx, ny, nz), dtype=np.float32)
    gridDims = ((nx + TPBX - 1) // TPBX,
                (ny + TPBY - 1) // TPBY,
                (nz + TPBZ - 1) // TPBZ
                )
    blockDims = (TPBX, TPBY, TPBZ)
    morph_kernel[gridDims, blockDims](d_f, d_x, d_y, d_z, t, C)
    return d_f.copy_to_host()
==================================================================
```

File format transformation

Since 3D objects are exporting, we store the result as PLY files in python. Then we used Blender to convert the file into FBX type. PLY contains faces, vertices and a variety of properties, including: color and transparency, surface normals, texture coordinates and data confidence values of the object while FBX contains models (faces and vertices), animation , texture and material. And FBX is supported by Unity which we will use to deploy the application.

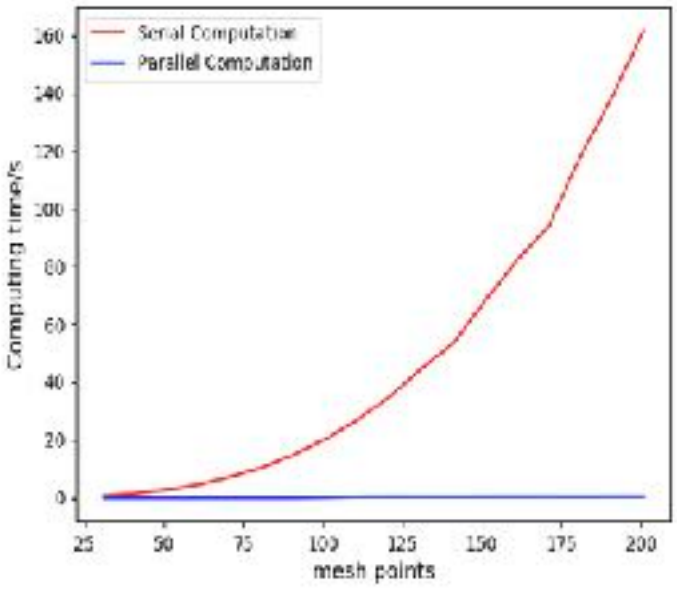
Deployment

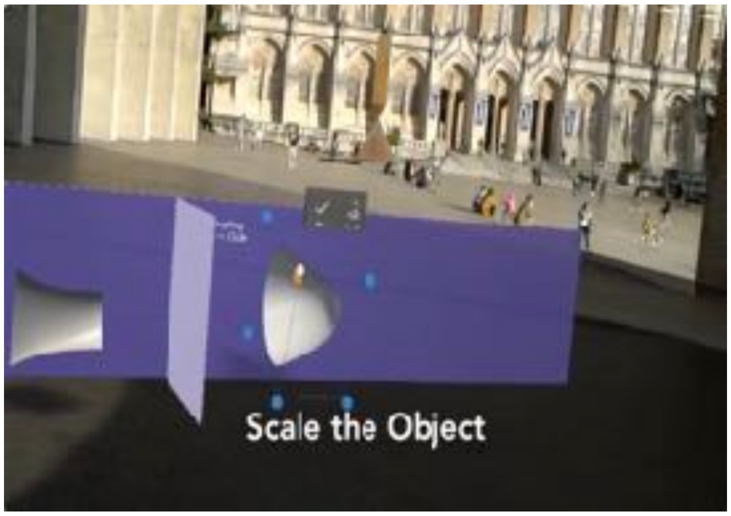
We showed all the result objects in HoloLens. In order to make it more interesting, we added interactive properties, for example, scaling, moving, rotating the object using your hand. All these must be programed in C# in Visual Studio.

Firstly we add hands detection so that the HoloLens knows when we rise our hands. Secondly, we add hand and gaze (pointer in HoloLens representing where we are staring at) manager so that further interactions can be performed (it will think we selected the object if it detects our hands hovering around/over the gaze). Thirdly, we add a box rounding the object indicating it is selected. Then, when the object is selected, it can be scaled, self-rotated or rotated so that users can inspect the object all around 360 degrees. Finally, we build a solution to Visual Studio and then deploy the solution as an application to HoloLens via WIFI.

Results

Speed-up: When we used 200 mesh points calculating the 2D morphing, a speed up of over 160 times was observed. (Serial is 165s and parallel is 1s) Then we tested it several times and obtained the following speed-up graph. (left) However, because of the high density of the grid points, to avoid crashing the machine, we just run a couple times and speed-up in 3D is approximately over 300 times (serial is 386s and parallel is 1s).

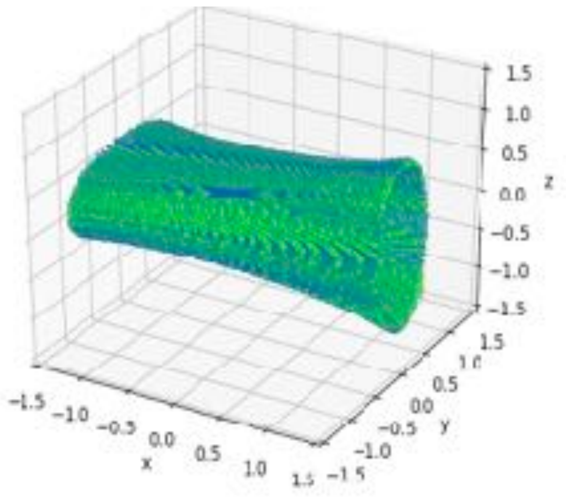


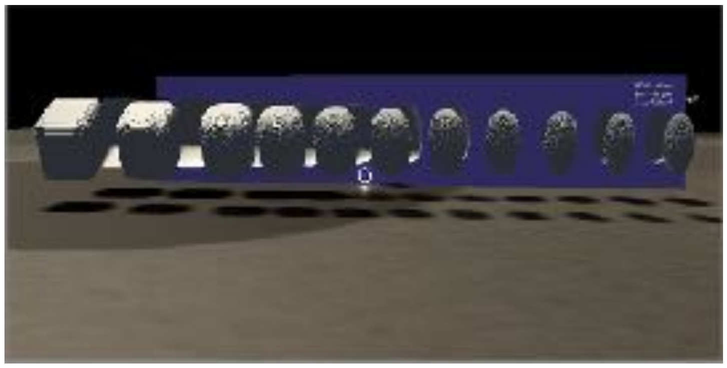


Picture above on the right is the scene of what the morphed objects look like in AR. (It was proudly recorded at the red square of University of Washington, Seattle) The selected object is the one we built to test out the app and it is generated from morphing a 2D circle to a smaller 2D ellipse.

==
Hardware:
>CPU: Intel Core i7-4700 HQ @ 2.6 GHz
>GPU: GeForce GTX 970M with 1280 CUDA cores.
>The clock rates specified by Nvidia are 924 MHz (+ Boost) for the GPU and 1250 MHz (5000 MHz effective) for the up to 6 GB of GDDR5 memory.
==

Morphing in 2D (left blow) and 3D (right below).





Obstacles and workarounds

1. Software version is a huge deal! If you fail to get the best fit version, you may encounter various problems/bugs that stop you from successfully deploying the app. We tried over 10 version combinations of Unity and Visual Studio. We are now confident with Unity 2017 1.0p5 and Visual Studio 2017 and HoloLens default version (do not upgrade).

2. About the theme of the project, initially, we tried to implement a voxel method on a 3D object (like Minecraft). However, just detecting whether a point from the point cloud of the object is in the box and 'light it up' (make visible) is an old-school voxelization method. Instead, with the suggestion of the professor, we decided to do something special instead of just voxelization. The idea of morphing came from professor and we implemented a parallel version of it. Another thing is that we didn't use functions with sharp turns, which didn't look good and didn't seem to be correct. We convinced ourselves that the point is to use CUDA to achieve significant speed up in morphing and illustrate it in AR environment.

3. We tried to make cartoons on Blender using embedded python environment. But we couldn't solve the problem that Blender cannot import python packages when we followed along tutorials. Then we were not able to build real-time app and cartoons on Blender, neither did parallel computation via CUDA. We gave up on this direction and turn to use Unity since Unity is the mainstream in developing 3D graphical games and videos, especially apps on HoloLens. But another problem is, we then have to use C#, which leads to problem#4 below.

4. We are all new to Unity environment and C#, with no experience in developing in 3D graphical gaming software. We searched through tons of tutorials and examples we could find to learn to add gesture and interactive properties to our AR app. With the help of Microsoft HoloLens course and Microsoft Unity open source applications, we successfully added those desired properties with the C# scripts embedded.

5. We tried to do much finer 3D models (larger grid point sets), however, our machine which runs parallel computation with CUDA is out of memory when we used over 1000 point in one directions. We decided a trade-off that we only picked 500 points and it still looked good. Besides, larger point set means larger file, which may cause time-out when deploying the app to HoloLens.

6. For 2D morphing, we used a circle morphs into an ellipse at the first place, which gives a bowl-like shape. Then we changed to two ellipses, one with the semi-major axis on x-axis and the other one with the semi-major axis on y-axis, which may be more interesting when seeing it in AR environment.

Future improvements / to-dos

1. Render a nicer background. (Need solid Unity modeling skills)

2. Figure out more complex functions. (Mathematica or other software that can generate functions for complex shape may be required. And have to figure out the sharp turns when building models)

3. More complex interactive properties. Like dynamically get an intermediate morphed shape with specific weight t by sliding your hand.

4. Then it rises the next improvement: real time (online) parallel model calculation and real-time AR exhibition. This may need to find a way to make python and Unity directly communicate, which is not likely right now said by the community of Unity developers. (If Unity has CUDA support in C#, then the whole process can be run in Unity only, which may make real-time possible.)

5. For now only scale and rotate are realized due to limited time. The ability to move the object will need further calculation on the position in HoloLens.

Lesson learnt

The first thing we are interested in is the idea of morphing, which is completely new to us . Professor showed us the linear combination of 2 functions may give rise to a more interesting shape, which would help better understanding if the shape is deploy in AR so that user can inspect into it and manipulate it.

Then we digged into HoloLens and tried to understand the structure of the pipeline on displaying digital objects in our reality. By tackling with Unity, we learnt that in order to make an interactive app, we must first let HoloLens detect our hands. Then use 3D position information to add size or position transition to the object depends on which type of operation the user performs (scale, self-rotating or rotate).

We also learnt the trick behind when plotting a surface of a 3D object and output the correct PLY file format with correct information (the generated meshed faces and vertices). Converting it into FBX format can make it readable to Unity, while FBX contains models (faces and vertices), animation , texture and material. This is the point we know what information does software need to read those formats and how does an object form in the software.

The main point of this class is parallel computation and we certainly make good use of it to achieve large speed-ups. With such practices, we are more familiar and confident with parallel computation in python using CUDA.

We also learnt if we are trying to make something big, something exciting, we need more skills and knowledges which can let our imagination fly. And cooperation of different people expertise in different skills or areas is also crucial.

Source code and files

Related GitHub repository: https://github.com/TianqiLi7398/ME599