**DICOM HoloLens Application**

*Step-by-step documentation by Martin Lehmann*

1. Transforming DICOM into .raw files
2. Loading .raw files into Unity3D
3. Manipulation of the volume-data-cube in unity
4. Using the HoloLens app

**Transforming DICOM into .raw files**

Unity3D cannot use the DICOM data format so we have to transform everything into a usable data format. We used Slicer3D for that, with the help of this program it is possible to save DICOM data into the .nrrd format, which will also produce a .raw, file. After unzipping the file, we can load the new .raw file into the Unity3D application. In the future, we will write our own script for this file transformation and not use Slicer3D.

**Loading .raw file into Unity3D**

Unity cannot load the created .raw file into our program so we had to write a loader script for that. The simple script will search a resource folder for .raw files based on the file name and then load it onto the game object. For this we created a simple cube, attached the loader script and after we activated the script we simply need to give the height, depth and width we need for our data, then start the application. In the future, it will be possible to read the height, depth and width automatically. The loader script will now create a volume-data file from our .raw file and save it into the resource folder. After that, it is important to deactivate the loader script so we can work with the volume-data.

Code example Loader:

public class Loader : MonoBehaviour {

public string path = @"Assets/";

public string filename = "skull";

public string extension = ".raw";

public int[] size = new int[3] { 512, 512, 512 }; //last value = number of DICOM files used to build the 3D object

public bool mipmap;

void Start() {

Color[] colors = LoadRAWFile(); //Load RGBA values of Data into a color array

Texture3D texture = new Texture3D (size[0], size[1], size[2], TextureFormat.Alpha8, mipmap);

//creat a 3D Texture with heigth, width and depth of size

texture.SetPixels (colors); //Set pixel RGBA value to color array values

texture.Apply (); // assign it to the material of the parent object

GetComponent<Renderer>().material.SetTexture ("\_Data", texture); // save it as an asset for re-use

#if UNITY\_EDITOR

AssetDatabase.CreateAsset(texture, path+filename+".asset");

#endif

}

**Manipulation of the volume-data-cube**

After loading the .raw file and creating the volume-data we can attach the volume-data file to our material, this material is the standard unity material and attached to the cube. The material uses a shader that makes it possible to see and work with the volume-data files. The shader has two main uses:

1. Render the volume-data onto the cube and
2. Give us options to change the used data.

Within the property block of the shader are all options we have to manipulate the volume-data-cube (slicing, brightness, quality).

Code example Shader

Shader "Custom/Ray Casting" {

Properties {

// the data cube

[NoScaleOffset] \_Data ("Data Texture", 3D) = "" {}

\_Axis ("Axes order", Vector) = (1, 2, 3)

\_TexFilling ("Data filling factors", Vector) = (1, 1, 1)

\_SliceAxis1Min ("Slice along axis X: min", Range(0,1)) = 0

\_SliceAxis1Max ("Slice along axis X: max", Range(0,1)) = 1

\_SliceAxis2Min ("Slice along axis Y: min", Range(0,1)) = 0

\_SliceAxis2Max ("Slice along axis Y: max", Range(0,1)) = 1

\_SliceAxis3Min ("Slice along axis Z: min", Range(0,1)) = 0

\_SliceAxis3Max ("Slice along axis Z: max", Range(0,1)) = 1

\_DataMin ("Data threshold: min", Range(0,1)) = 0

\_DataMax ("Data threshold: max", Range(0,1)) = 1

\_StretchPower ("Data stretch power", Range(0.1,3)) = 1

\_NormPerStep ("Intensity normalization per step", Float) = 1

\_NormPerRay ("Intensity normalization per ray" , Float) = 1

\_Steps ("Max number of steps", Range(1,1024)) = 128

}

Scripts that directly work on the cube and not the volume-data provide other options of manipulation. Move, zoom and rotate the cube by manipulating the virtual camera position, scaling of the cube and rotating the cube along his axis (from the viewpoint of the virtual camera). The Xbox One wireless controller (every input shown in the controller mapping documentation) controls every option.

Code example Xbox One wireless controller

//Movement speed

public float MoveHorizontalSpeed = 2.0f;

public float MoveVerticalSpeed = 2.0f;

public void translateRotateScale()

{

//Moving the Object with the left stick

float moveHorizontal = MoveHorizontalSpeed \* controllerInput.GetAxisLeftThußstickX();

float moveVertical = MoveVerticalSpeed \* controllerInput.GetAxisLeftThumbstickY();

this.transform.Translate(moveHorizontal, moveVertical, 0.0f);

//Rotate the Object with the Left and Right Shoulder Buttons,

clockwise and counterclockwise

float rotateAroundY = RotateAroundYSpeed \*

(controllerInput.GetAxisRightTrigger() - controllerInput.GetAxisLeftTrigger());

//Rotate the Object with right stick

float rotateAroundX = RotateAroundXSpeed \* controllerInput.GetAxisRightThumbstickY();

float rotateAroundZ = RotateAroundZSpeed \* controllerInput.GetAxisRightThumbstickX();

this.transform.Rotate(rotateAroundX, -rotateAroundY, rotateAroundZ);

**Using the HoloLens Application**

Using the application with Microsoft HoloLens is simple. After creating and loading, the volume-data onto the cube go to:

* File Menu, select Build settings (Make sure to activate your scene in the Scenes In Build window).
* Select Windows Store in the Platform menu and
  + switch SDK to Universal 10,
  + Target device to any device,
  + UWP Build Type to XAML,
  + UWP SDK to latest installed,
  + Build and Run on to local machine.
  + Check the Copy References and Unity C# Project checkbox
* Then Build the Project.

A window to create a file will pop up, create a new folder called App and select it. Wait for the program to build the file then open it. Open the App folder select the .sln file

and open it. This will start Visual Studio. In VS on the top menu there are 3 drop down menus, in the first select Release, in the second x86 and in the last Device. After that go to Debug and select Start without Debugging. Now the app will start on your HoloLens.

A well-known error in VS is preventing the app from starting sometimes. If an error massage pops up, press ok and then select Start without Debugging again this should correct the error and start the app on HoloLens.