CSE471 Introduction to Computer Graphics (fall 2019)
Due date: Dec 8, 2019, 11:59 pm.

Assignment 4: Volume Rendering

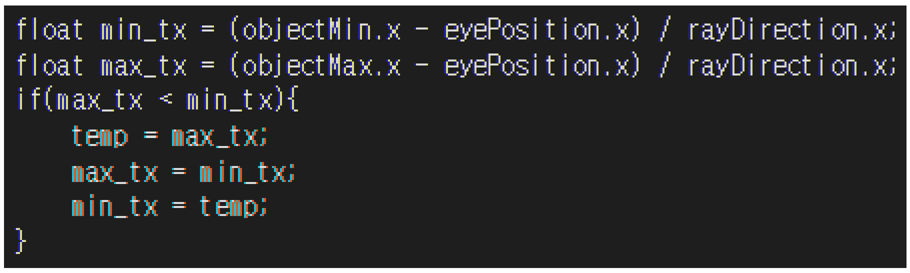
20131218
Park KeeHun

1. Introduction

In this assignment, I implement direct volume rendering, and specifically, I implement a ray casting algorithm using alpha compositing. I just modify the code of fragment shader, *volumeRendering.frag,* and realize 3 parts. First part is finding intersect points of ray and object, entry and exit point. Second part is implementing maximum intensity projection (MIP), and third part is implementing alpha compositing.

2. Method

2-1. Find ray-cube intersect point

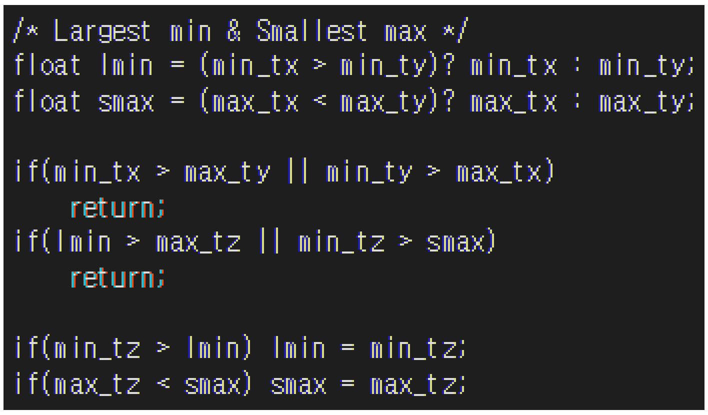


```
float min_tx = (objectMin.x - eyePosition.x) / rayDirection.x;
float max_tx = (objectMax.x - eyePosition.x) / rayDirection.x;
if(max_tx < min_tx){
    temp = max_tx;
    max_tx = min_tx;
    min_tx = temp;
}
```

I calculate 6 values, min_tx, max_tx, min_ty, max_ty, min_tz, max_tz, which are range that ray penetrates the object per 3 axis. In volumeRendering.frag, I bring several values, objectMin, objectMax, and eyePosition. And rayDirection is already calculated. So, by using these information, I can calculate intersecting range with below equation,

objectPoint = eyePosition + rayDirection * t   (t is scalar value)

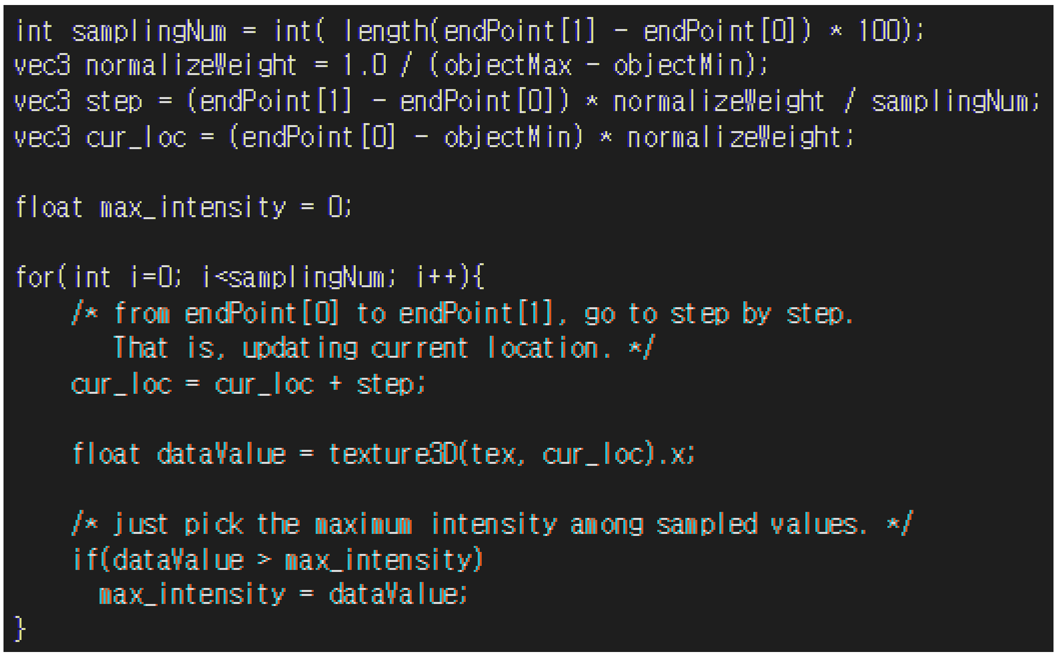
And I also consider the case that ray penetrates the object reversely. In this case, ray first meet the max_plane before min_plane, so min_tx becomes larger then max_tx. To prevent these error, I must change two values each other.



```
/* Largest min & Smallest max */
float lmin = (min_tx > min_ty)? min_tx : min_ty;
float smax = (max_tx < max_ty)? max_tx : max_ty;

if(min_tx > max_ty || min_ty > max_tx)
    return;
if(lmin > max_tz || min_tz > smax)
    return;

if(min_tz > lmin) lmin = min_tz;
if(max_tz < smax) smax = max_tz;
```

After taking 6 values, I should find exact range that ray penetrates only the object. Each value means not a side of cube but plane, so if we shoot the ray from eye position to object, it can penetrate 4 plane. Above code is distinguish what plane is side of cube.
lmin is the largest minimum and smax is the smallest maximum, and by using previous equation again, I can find entry and exit points.

2-2. Maximum intensity projection (MIP)

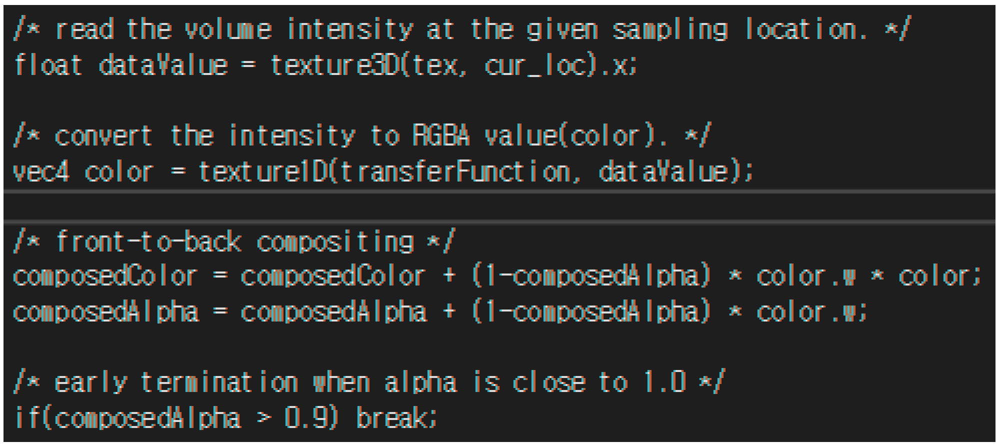


```
int samplingNum = int( length(endPoint[1] - endPoint[0]) * 100);
vec3 normalizeWeight = 1.0 / (objectMax - objectMin);
vec3 step = (endPoint[1] - endPoint[0]) * normalizeWeight / samplingNum;
vec3 cur_loc = (endPoint[0] - objectMin) * normalizeWeight;

float max_intensity = 0;

for(int i=0; i<samplingNum; i++){
    /* from endPoint[0] to endPoint[1], go to step by step.
       That is, updating current location. */
    cur_loc = cur_loc + step;

    float dataValue = texture3D(tex, cur_loc).x;

    /* just pick the maximum intensity among sampled values. */
    if(dataValue > max_intensity)
      max_intensity = dataValue;
}
```

To implement MIP, I realize walking from endPoint[0] to endPoint[1] with step by step. normalizeWeight is needed for values to be suitable for the object size. So after, to standardize values to the object size, I just multiplying normalizeWeight.
One step is just dividing length between endPoint[0] and endPoint[1] by sampling number. cur_loc is my current location, so in for loop, I should update this value per every step. Walking from endPoint[0] to endPoint[1] is implemented by using for loop with step.

And in maximum intensity projection(MIP), I just pick the maximum intensity. So, before entering into loop, initialize max_intensity as 0, then in loop I update it.

2-3. Alpha composition



```
/* read the volume intensity at the given sampling location. */
float dataValue = texture3D(tex, cur_loc).x;

/* convert the intensity to RGBA value(color). */
vec4 color = texture1D(transferFunction, dataValue);

/* front-to-back compositing */
composedColor = composedColor + (1-composedAlpha) * color.w * color;
composedAlpha = composedAlpha + (1-composedAlpha) * color.w;

/* early termination when alpha is close to 1.0 */
if(composedAlpha > 0.9) break;
```

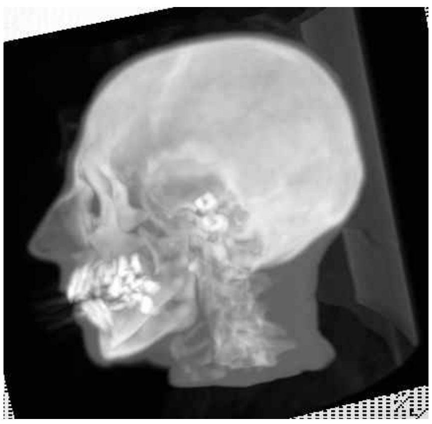
Except inner part of for loop, it is same as above MIP codes. To implement alpha composition, I should take values from transferFunction, which can be controlled by user mouse, then convert it into RGBA value, vec4 color.
Because I should use front-to-back compositing, I update two values, composeColor and composedAlpha, with below equation,

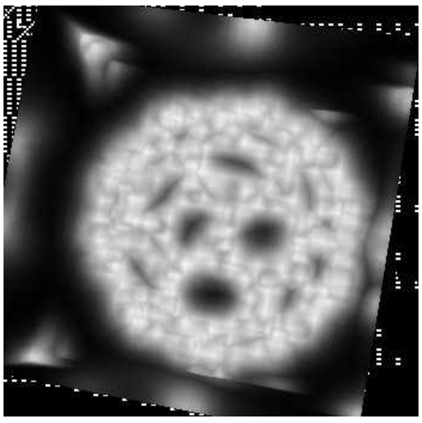
$$\boxed{\begin{aligned} C'_{i+1} &= C'_i + (1 - A'_i)A_i c_i \\ A'_{i+1} &= A'_i + (1 - A'_i)A_i \end{aligned}}$$

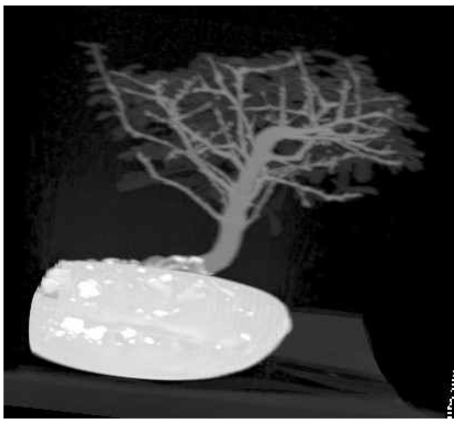
And I also have to use early termination if alpha value is close to 1.0. So, I set if-break in for loop, when composedAlpha becomes larger than 0.9.

3. Result

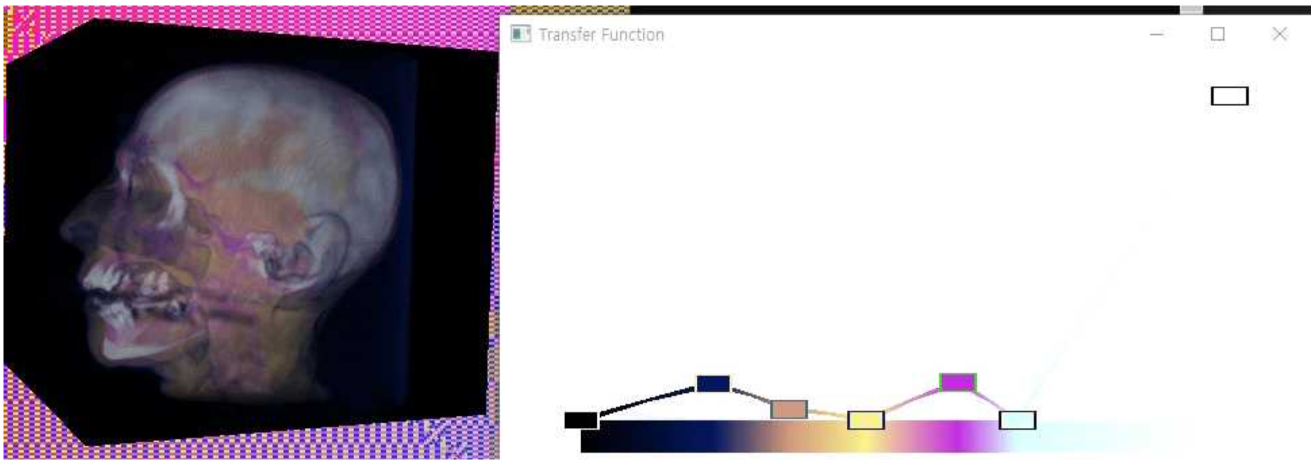
3-1. Maximum intensity projection (MIP)


CT head

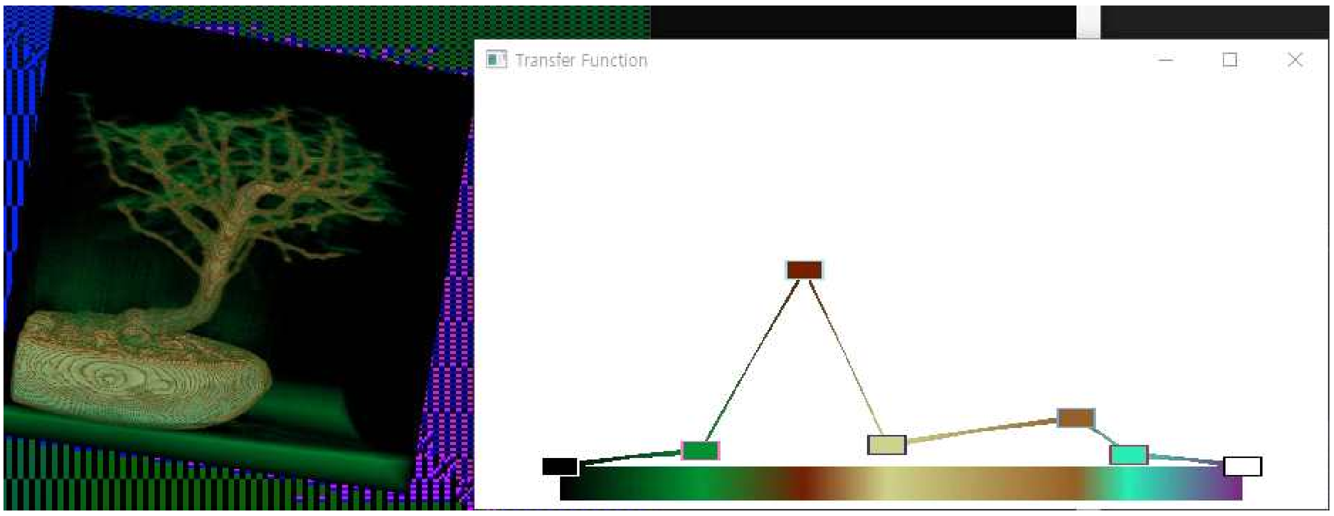

Bucky


Bonsai

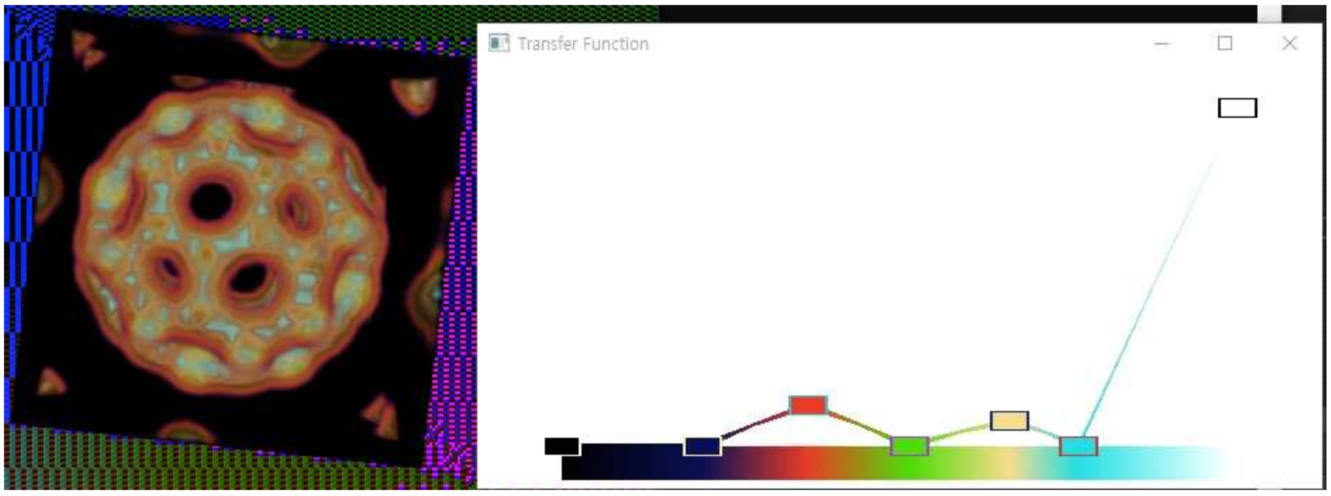
3-2. Alpha composition (with Transfer Function)



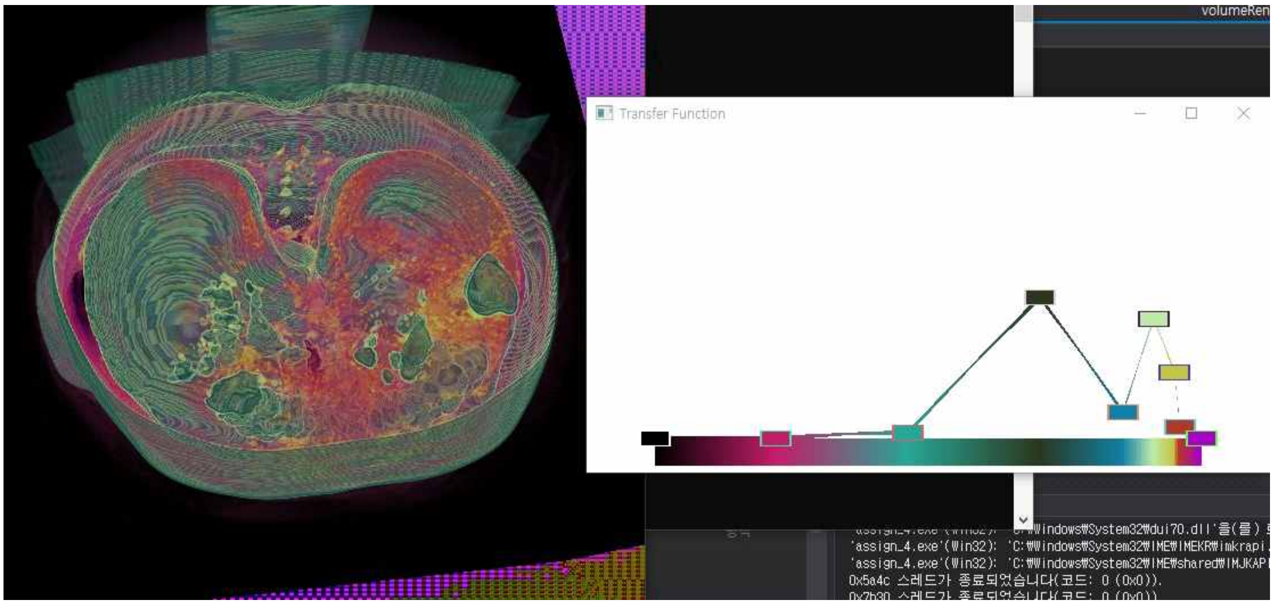
CT head



Bonsai

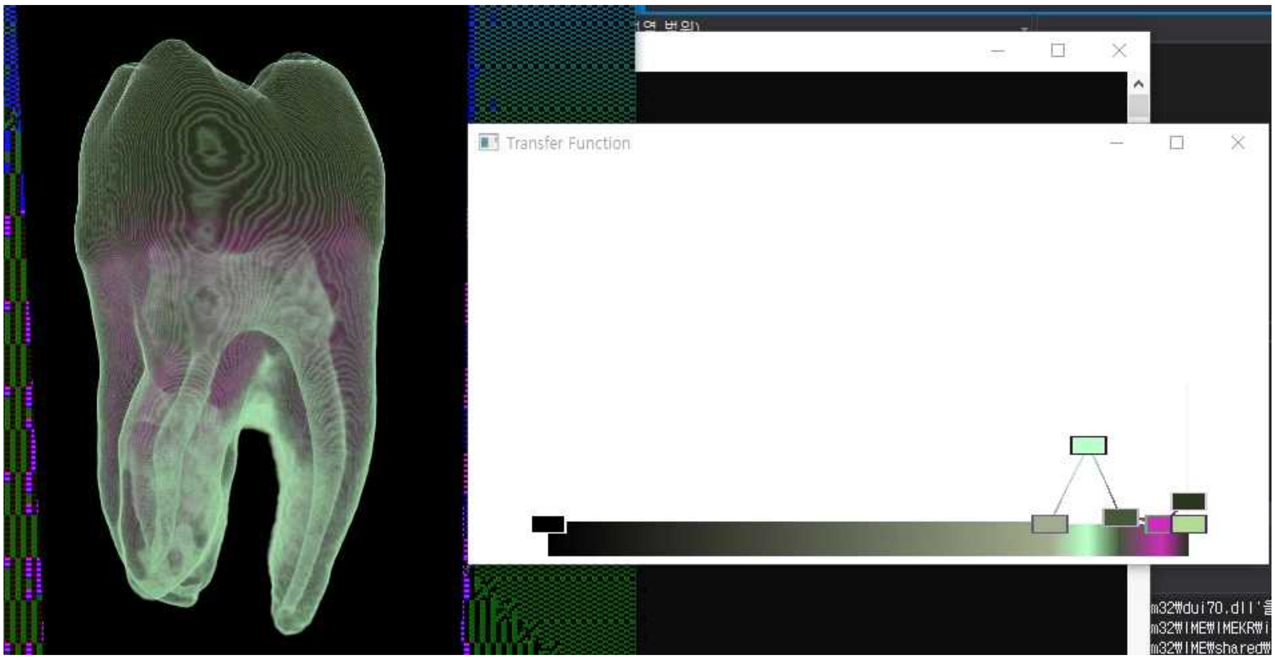


Bucky



```
'assign_4.exe'(Win32): 'C:₩Windows₩System32₩dui70.dll'을(를)
'assign_4.exe'(Win32): 'C:₩Windows₩System32₩IME₩IMEKR₩imkrapi.
'assign_4.exe'(Win32): 'C:₩Windows₩System32₩IME₩shared₩IMJKAP
0x5a4c 스레드가 종료되었습니다(코드: 0 (0x0)).
0x7b30 스레드가 종료되었습니다(코드: 0 (0x0)).
```

Lung



```
m32₩dui70.dll'을
m32₩IME₩IMEKR₩i
m32₩IME₩shared₩
```

Tooth

4. Conclusion

Through this assignment implementation, I can learn and understand about volume rendering. Although I have experience to deal with fragment shader in assignment 2, it is hard to understand and implement this assignment. However, it is very exciting that image of volume rendering is very realistic and even beautiful. Especially, image of CT_head, lung, or tooth are familiar in hospital, so it is more marvelous for me. So, I think that this assignment can be good experience for me.