```
//--------------------------------------------------------------------------------
//
// Geometry shader for Voxelization
// @ Input: vertices of a triangle
// @ Output: duplicate of triangles for slice
//
//--------------------------------------------------------------------------------
void gs_main(triangle VS_TEX_OUT input[3], inout TriangleStream<GS_OUT> triStream)
{
    // Find the max and min of the depth
    float distMax=max(max(input[0].posV.z,input[1].posV.z),input[2].posV.z);
    float distMin=min(min(input[0].posV.z,input[1].posV.z),input[2].posV.z);

    // Find the range of the valid depth
    float2 start_end=floor(float2(distMax,distMin)/gSliceValue);


    float4x4 ClipMatrix = gProj;
    float2   Near_far   = float2(0,0);

    GS_OUT output;

    // Duplicate the triangles according to the intersection of the depth slices
    for(int g=start_end.y; g<=start_end.x; g++)
    {
        // Check whether the depth is inside the range
        if(g>0 && g<gMaxDepth)
        {
            output.RTIndex =g;

            float2 clip = float2(g,g+1)*gSliceValue+float2(-gSliceValue,gSliceValue);

            ClipMatrix._33 = clip.y/(clip.y-clip.x);
            ClipMatrix._43 = -clip.x*clip.y/(clip.y-clip.x);

            for(int v=0; v<3; v++)
            {
                output.Pos = mul(input[v].posV,ClipMatrix);
                output.norL=input[v].norL;
                triStream.Append( output );
            }
            triStream.RestartStrip( );
        }
    }
}


//--------------------------------------------------------------------------------
//
// Two-surface refraction  pixel shader
// @ Input: voxelization result
//
//--------------------------------------------------------------------------------
float4 ps_main(VS_OUT pIn)
{
    // Compute the view direction in world
    float3 viewVec = normalize(  pIn.posW -gEyePosW);

    // Set up the step distance
    float3 StepSize=0.001953125f;
```

```
    // Set up tracking direction
    float3 dir=viewVec;

    // Set up the tracking position
    float3 pos=pIn.posL;

    // Preset the air refraction index
    float Index=1.14f;
    float InvIndex=1/Index;

    bool InObj=false;
    int InCount=0;

    float3 lastnormal=float3(0,0,0);
    float thick = 0;

    float3 move;
    bool hit=false;
    float4 normal= pIn.normal;

    // Transform the tracking direction into voxelization coordinate
    move=mul(float4(dir , 0.0f) ,gVolume);
    move.y=move.y*-1;
    move=normalize(move);

    // Compute the first refraction
    dir=refract(dir,normal,InvIndex,Index);

    // Go through the object to find the second refraction
    for(int i=0;i<MAX_STEP;i++)
    {
        normal=loadData(move,pos);
        hit=false;
        if(dot(normal.rgb,lastnormal.rgb)>0.75f)
            hit=true;
        if(any(normal.rgb) && !hit)
        {
            lastnormal=normal;
            if(dot(normal.xyz,-dir.xyz)<0){
                InObj=false;
            }else{
                InObj=true;
            }
        }
        if(InObj)
            thick++;
        // Advect the tracking position
        pos=StepSize*move+pos;
        if(pos.z<0 || pos.z>1.0f || pos.x > 1.0f || pos.y > 1.0f || pos.x <0.0f  || pos.y < 0.0f ){
            break;
        }
    }

    // Compute the second refraction
    dir=refract(dir,lastnormal,InvIndex,Index);

    float4 reflectedColor = CubeMapSample(dir );
    return reflectedColor ;
}

//*********************************************************************************
```

```
//
// Multiple-surface refraction  pixel shader
// @ Input: voxelization result
//
//*************************************************************************************
float4 ps_main(VS_OUT pIn)
{
    // Compute the view direction in world
    float3 viewVec = normalize(  pIn.posW -gEyePosW);

    // Set up the step distance
    float3 StepSize=0.001953125f;

    // Set up tracking direction
    float3 dir=viewVec;

    // Set up the tracking position
    float3 pos=pIn.posL;

    // Preset the air refraction index
    float Index=1.14f;
    float InvIndex=1/Index;

    bool InObj=false;
    int InCount=0;
    float3 lastnormal=float3(0,0,0);
    float3 move;
    bool hit=false;
    float4 normal;

    // Transform the tracking direction into voxelization coordinate
    move=mul(float4(dir , 0.0f) ,gVolume);
    move.y=move.y*-1;
    move=normalize(move);

    for(int i=0;i<MAX_STEP;i++)
    {
        normal=loadData(move,pos);
        hit=false;

        // Check the similarity of refraction normal
        if(dot(normal.rgb,lastnormal.rgb)>0.75f)
            hit=true;

        // Not similar and then do the refraction
        if(any(normal.rgb) && !hit)
        {
            lastnormal=normal;

            // Check the front or back face entrance
            if(dot(normal.xyz,-dir.xyz)<0){
                dir=refract(dir,-1*normal.xyz,Index,InvIndex);
                InObj=false;
            }else{
                dir=refract(dir,normal.xyz,InvIndex,Index);
                InObj=true;
            }
            dir=normalize(dir);

            move=mul(float4(dir , 0.0f) ,gVolume);
            move.y=move.y*-1;
```

```
            move=normalize(move);
        }

        // Advect the tracking position
        pos=StepSize*move+pos;
        if(pos.z<0 || pos.z>1.0f || pos.x > 1.0f || pos.y > 1.0f || pos.x <0.0f  || pos.y < 0.0f ){
            break;
        }
    }

    float4 reflectedColor =  CubeMapSample(dir );

    return reflectedColor ;
}
```