Project : Kirby Discovery (3D 팀 프로젝트 최성희) _ 검광효과

코드 목적 :

검이 이동한 궤적에 따라 잔상을 보여주기 위하여 구현했습니다.

코드 설명 :

1. Effect_Trail 클래스는 검광 효과를 사용하고자 하는 객체가 소유하도록 구현했습니다.
2. 해당 객체는 Effect_Trail 클래스의 Add_TrailPoint() 함수를 호출하여 월드 스페이스 상의 두 점 위치를 넘겨줍니다.
3. 두 점을 넘겨받았을 때, 궤적을 보관하고 있는 컨테이너가 비어있으면 넘겨받은 좌표로 컨테이너를 채워줍니다.
4. 컨테이너가 비어있지 않으면 컨테이너 가장 앞쪽에 추가하고, 가장 뒤에 있는 좌표를 삭제해줍니다.
5. XMVectorCatmullRom() 보간 함수를 사용하여 점의 위치를 다시 갱신했습니다. (부드러운 곡선을 만들어주기 위하여 사용했습니다.)
6. Trail_Update() 함수에서는 CVIBuffer_Trail 클래스의 Update()를 호출하여 갱신될 정점의 위치 정보를 넘겨줍니다.
7. 넘겨받은 월드 상의 위치정보로 버퍼의 로컬위치를 변환시켜 그려지게 했습니다.

```
void CEffect_Trail::Add_TrailPoint(_float4 fTop, _float4 fBottom)
{
    if (m_pTrailTopPoint.size() == 0)
    {
        for (_uint i = 0; i < m_iNumInstance + 1; ++i)
        {
            m_pTrailTopPoint.push_back(fTop);
            m_pTrailBottomPoint.push_back(fBottom);
        }
    }
    else
    {
        auto& iterbegin = m_pTrailTopPoint.begin();
        m_pTrailTopPoint.insert(iterbegin, fTop);
        auto& iterend = m_pTrailTopPoint.end();
        m_pTrailTopPoint.erase(--iterend);


        auto& iterbegin1 = m_pTrailBottomPoint.begin();
        m_pTrailBottomPoint.insert(iterbegin1, fBottom);
        auto& iterend1 = m_pTrailBottomPoint.end();
        m_pTrailBottomPoint.erase(--iterend1);

        CatmullRom();
    }
    CreateTrail();
    Trail_Update();
}
```

```
void CEffect_Trail::CatmullRom()
{
    _uint i = 0;
    while (true)
    {
        _vector vTrailTop = XMVectorCatmullRom(XMLoadFloat4(&m_pTrailTopPoint[i]),
            XMLoadFloat4(&m_pTrailTopPoint[i + 1]), XMLoadFloat4(&m_pTrailTopPoint[i + 2]),
            XMLoadFloat4(&m_pTrailTopPoint[i + 3]), 0.1f);

        XMStoreFloat4(&m_pTrailTopPoint[i + 1], vTrailTop);

        _vector vTrailBottom = XMVectorCatmullRom(XMLoadFloat4(&m_pTrailBottomPoint[i]),
            XMLoadFloat4(&m_pTrailBottomPoint[i + 1]), XMLoadFloat4(&m_pTrailBottomPoint[i + 2]),
            XMLoadFloat4(&m_pTrailBottomPoint[i + 3]), 0.1f);

        XMStoreFloat4(&m_pTrailBottomPoint[i + 1], vTrailBottom);
        ++i;

        if (i + 3 == m_iNumInstance + 1)
            break;
    }
}
```

Project : Kirby Discovery (3D 팀 프로젝트 최성희) _ SSD

코드 목적 :

두 개의 폴리곤이 같은 깊이 값을 가지고 있으면 z-fighting이 일어나 깜박거리는 현상이 발생하는데 이 현상을 해결하고, 오브젝트 굴곡에 따라 원하는 텍스쳐를 붙이고자 구현했습니다.

코드 설명 :

1. 랜더링 파이프라인의 Deferred Rendering 시스템에서 오브젝트의 깊이 값이 기록되어있는 깊이 랜더타겟과 법선벡터가 기록되어있는 노말 랜더타겟을 이용합니다.

2. 따라서, 모든 오브젝트가 전부 그려진 후 마지막으로 그려지도록 했습니다. (SSD는 Cube 버퍼를 사용하여 그립니다)

3. 붙여지는 오브젝트의 법선벡터와 붙이고자 하는 오브젝트의 방향을 내적 하여 특정 각도 이상이면 그려지지 않도록 했습니다.

4. 깊이 랜더타겟을 이용하여 상자(Cube 버퍼) 밖에 위치한 경우 해당 픽셀은 그리지 않도록 했습니다.

5. 데칼 박스의 버퍼가 -0.5 ~ 0.5 사이이므로 0.5를 더해줘서 UV 좌표를 만들고 해당 텍스쳐를 그려주었습니다.

6. 블러 마스크를 사용하여 SSD에도 블러가 적용되도록 했습니다.

```
PS_OUT_BLUR PS_SSD_Blur(PS_IN_DECAL In)
{
    PS_OUT_BLUR     Out = (PS_OUT_BLUR)0;

    /* Decal Box Renderering */
    float2 vTexUV;
    vTexUV.x = (In.vProjPos.x / In.vProjPos.w) * 0.5f + 0.5f;
    vTexUV.y = (In.vProjPos.y / In.vProjPos.w) * -0.5f + 0.5f;
    vector vDepthDesc = g_DepthTexture.Sample(DefaultSampler, vTexUV);
    vector vNormalDesc = g_NormalTexture.Sample(DefaultSampler, vTexUV);

    clip(dot(vNormalDesc, In.vDecalDir) - cos(radians(60.f)));

    if (vDepthDesc.z == 1.f)
        discard;

    float fViewZ = vDepthDesc.x * g_fFar;
    vector vDecalLocalPos;
    vDecalLocalPos.x = (vTexUV.x * 2.f - 1.f) * fViewZ;
    vDecalLocalPos.y = (vTexUV.y * -2.f + 1.f) * fViewZ;
    vDecalLocalPos.z = vDepthDesc.y * fViewZ;
    vDecalLocalPos.w = 1.f * fViewZ;

    /* Object in Decal Local Space */
    vDecalLocalPos = mul(vDecalLocalPos, g_ProjMatrixInv);
    vDecalLocalPos = mul(vDecalLocalPos, g_ViewMatrixInv);
    vDecalLocalPos = mul(vDecalLocalPos, g_WorldMatrixInv);

    clip(0.5 - abs(vDecalLocalPos.xyz));

    float2 decalUV;
    decalUV.x = vDecalLocalPos.x + 0.5f;
    decalUV.y = vDecalLocalPos.z + 0.5f;
    vector vDecalDesc = g_DecalTexture.Sample(DefaultSampler, decalUV);

    vDecalDesc = vector(vDecalDesc.rgb * g_vColorMul + g_vColorAdd, vDecalDesc.r * g_vColorAlpha);
    vDecalDesc.a = saturate(vDecalDesc.a);

    if(vDecalDesc.a <= 0.f)
        discard;

    Out.vColor = vDecalDesc;
    Out.vBlurMask = vector(1.f, g_fBlurPower, 1.f, 1.f);

    return Out;
}
```

Project : Kirby Discovery (3D 팀 프로젝트 최성희) _ Lim

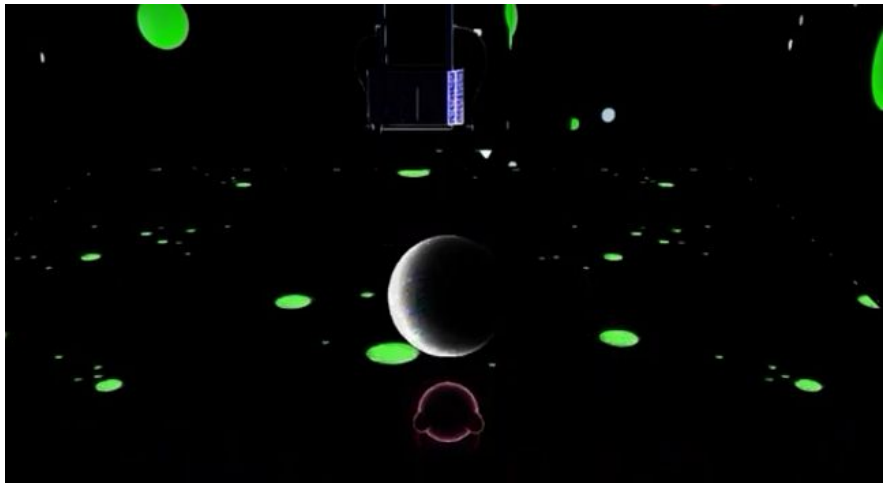
코드 목적 :

오브젝트의 외곽선에 색상변화를 주어 역광에서 빛이 비춰지는 효과를 나타내기 위해 사용했습니다.

코드 설명 :

1. 카메라의 방향과 오브젝트의 Normal벡터의 각도차를 이용했습니다.
2. 카메라가 바라보는 방향에 가장 큰 영향을 받으며 각도 차가 커질수록 색상을 다르게 표현했습니다.
3. 색상은 기본적으로 본인 Diffuse 색상을 적용하여 같은 객체이더라도 부위에 따라 자연스러운 색상변화를 주었습니다.
4. 색상과 외곽선 두께를 조절하여 오브젝트에 적용했습니다.

```
PS_OUT_RIM PS_MAIN_ONLY_RIM(PS_IN_RIM In)
{
    PS_OUT_RIM Out = (PS_OUT_RIM) 0;

    vector vNormal = Unpack_Normal(g_NormalTexture.Sample(WrapLinearSampler, In.vTexUV));

    float3x3 TBN = transpose(float3x3(In.vTangent.xyz, In.vBiNormal.xyz, In.vNormal.xyz));

    vNormal = vector(mul(TBN, normalize(vNormal.xyz)), 0.f);

    Out.vDepth = vector(In.vProjPos.w / g_fFar, In.vProjPos.z / In.vProjPos.w, 1.f, 0.f);
    Out.vVelocity.xy = (In.vProjPos.xy / In.vProjPos.w - In.vOldPos.xy / In.vOldPos.w);
    Out.vVelocity.z = 0.5f;
    Out.vVelocity.w = In.vProjPos.z / In.vProjPos.w;

    float fRim = saturate(dot(normalize(vNormal), normalize(vector(In.vViewPos.xyz, 0))));
    Out.vRim = pow(1.f - clamp(fRim, 0.f, 1.f), RimWidth) * Rim_Color ;
    Out.vRim += saturate(-fRim * 1.f * In.fAdd_Rim);

    return Out;
}
```



Project : Kirby Discovery (3D 팀 프로젝트 최성희) _ 왜곡효과

코드 목적 :

흐르는 물과 이펙트 사용 시 왜곡 효과를 주기 위해 구현했습니다.

코드 설명 :

1. 왜곡이 적용되어야 할 객체로부터 왜곡 텍스쳐를 받아와 왜곡 랜더타겟에 그려주었습니다.
2. 랜더타겟을 이용하여 왜곡이 필요한 곳에 부분적으로 적용했습니다.
3. 왜곡 텍스쳐의 색상 값으로 오리지날 이미지의 UV 값에 변화를 주어 왜곡 효과를 표현했습니다.

```
PS_OUT PS_MAIN_Refract(PS_IN In)
{
    PS_OUT Out = (PS_OUT) 0;

    vector vRefractDesc = g_RefractTexture.Sample(DefaultSampler, In.vTexUV);
    float2 UV = In.vTexUV + vRefractDesc.xy * 0.05f;
    Out.vColor = g_DiffuseTexture.Sample(BlurSampler, UV);

    return Out;
}
```

Project : Kirby Discovery (3D 팀 프로젝트 최성희) _ Fog

코드 목적 :

눈보라로 인한 안개효과를 표현하고자 했습니다. 또한, Stage에 따라 안개위치, 세기 등 다르게 주고자 했습니다.

코드 설명 :

1. 해당 코드는 HLSL프로그래밍 책을 참고했습니다.
2. 각 스테이지의 특성에 따라 카메라와의 거리를 기준으로 안개의 시작점, 감쇄되는 정도, 짙은 정도를 다르게 적용했습니다.
3. 필요에 따라 같은 맵에서도 트리거를 이용하여 장면마다 안개의 정도를 다르게 적용했습니다.
4. 이때, 안개 수치를 보간하여 자연스럽게 짙어지거나 옅어지도록 했습니다.
5. 스카이박스 처럼 깊이 값이 없는 곳에도 안개 색상과 보간하여 자연스럽게 보이도록 했습니다.

```
float3 ApplyFog(float3 originalColor, float eyePosY, float3 eyeToPixel)
{
    /* 카메라와의 거리 Depth 값*/
    float pixelDist = length(eyeToPixel);
    /* 카메라에서부터 방향벡터 */
    float3 eyeToPixelNorm = eyeToPixel / pixelDist;

    //픽셀 거리에 대해 안개 시작지점 계산 //FogStartDist : 안개 지점에서 카메라까지의 거리
    /* 안개 시작 지점에서 해당공간이 얼마나 떨어져있는지를 나타냄 */
    float fogDist = max(pixelDist - FogStartDepth, 0.f);

    //안개 세기에 대해 거리 계산
    /* 높이 소멸값 : 클수록 사라지는 높이가 낮아진다.*/
    float fogHeightDensityAtViewer = exp(-FogHeightFalloff * eyePosY);
    float fogDistInt = fogDist * fogHeightDensityAtViewer;

    //안개 세기에 대한 높이 계산
    float eyeToPixelY = eyeToPixel.y * (fogDist / pixelDist);
    float t = FogHeightFalloff * eyeToPixelY;
    const float thresholdT = 0.01f;
    float fogHeightInt = abs(t) > thresholdT ? (1.f - exp(-t)) / t : 1.f;

    //위 계산값을 합해 최종인수 계산
    float fogFinalFactor = exp(-FogGlobalDensity * fogDistInt * fogHeightInt);

    //태양 하이라이트 계산 및 안개 색상
    float sunHighlightFactor = saturate(dot(eyeToPixelNorm, FogSunDir));

    sunHighlightFactor = pow(sunHighlightFactor, 8.0);
    float3 fogFinalColor = lerp(FogColor, FogHighlightColor, sunHighlightFactor);

    return lerp(fogFinalColor, originalColor, fogFinalFactor);
}
```

Project : Kirby Discovery (3D 팀 프로젝트 최성희) _ 그림자

코드 목적 :

광원 방향에 따른 물체의 그림자를 만들고자 구현했습니다. 플레이어나 몬스터의 경우 다른 오브젝트의 그림자가 있더라도 더 진하게 그리도록 했습니다.

코드 설명 :

1. 그림자의 경우 카메라를 광원의 위치에 두고 물체들을 그린 후 깊이 값을 기록합니다. (랜더러에서 **ShadowRender()** 를 호출하여 그립니다.)

2. 현재의 깊이값과 그립자맵의 깊이값을 비교하여 깊이가 더 크면 그림자를 그렸습니다.

3. 광원의 위치는 고정으로 하여 사용하였고, 해당 맵의 섹션이 바뀌거나 장소가 바뀔경우 광원 위치와 투영행렬 값을 조절해 주었습니다.

4. AnimModel의 경우 주변 건물이나 오브젝트에 의해 그림자가 가려질 경우
더 진하게 그림자를 입혀 가려지지 않도록 하였습니다.

5. Sword같은 Item 장착시에도 다른 오브젝트의 그림자로부터 가려지지 않도록 적용했습니다.

```
HRESULT CKirby::ShadowRender()
{
    CTransform* pTransform = nullptr;
    CModel* pModel = nullptr;

    if (m_pItem && CItem::IT_EAT == m_pItem->Get_ItemType())
    {
        pTransform = static_cast<CTransform*>(m_pItem->Get_Component(COM_TRANSFORM));
        pModel = m_pItem->Get_ItemModel();
    }
    else
    {
        pTransform = m_pTransform;
        pModel = m_pModel;
    }

    FAILED_CHECK(__super::ShadowRender());
    FAILED_CHECK(pTransform->Bind_AllOnShader(m_pShader));

    _uint iNumMateyrials = m_pModel->Get_NumMaterials();

    for (_uint i = 0; i < iNumMateyrials; ++i)
    {
        pModel->Render(m_pShader, VTXANIMMODEL_ONLY_SHADOW, i, "g_BoneMatrices");
    }

    return S_OK;
}
```

```
PS_OUT_SHADOW PS_MAIN_SHADOW(PS_IN_SHADOW In)
{
    PS_OUT_SHADOW Out = (PS_OUT_SHADOW) 0;

    Out.vDepth = float4(0.f, In.vClipPos.z / In.vClipPos.w, 0.f, 1.f);

    return Out;
}
```

```
/* Shadow */
vector ShadowDepth = g_ShadowTexture.Sample(DefaultSampler, vTexUV);
vector AnimShadowDepth = g_AnimShadowTexture.Sample(DefaultSampler, vTexUV);
if (ShadowDepth.g < CurrentDepth - g_ShadowBias)
{
    rgb.rg = rgb.rg * 0.85f;
    rgb.b = rgb.b * 0.95f;

    if (AnimShadowDepth.g != 0.f)
    {
        rgb.rgb = rgb.rgb * 0.75f;
    }
}
```