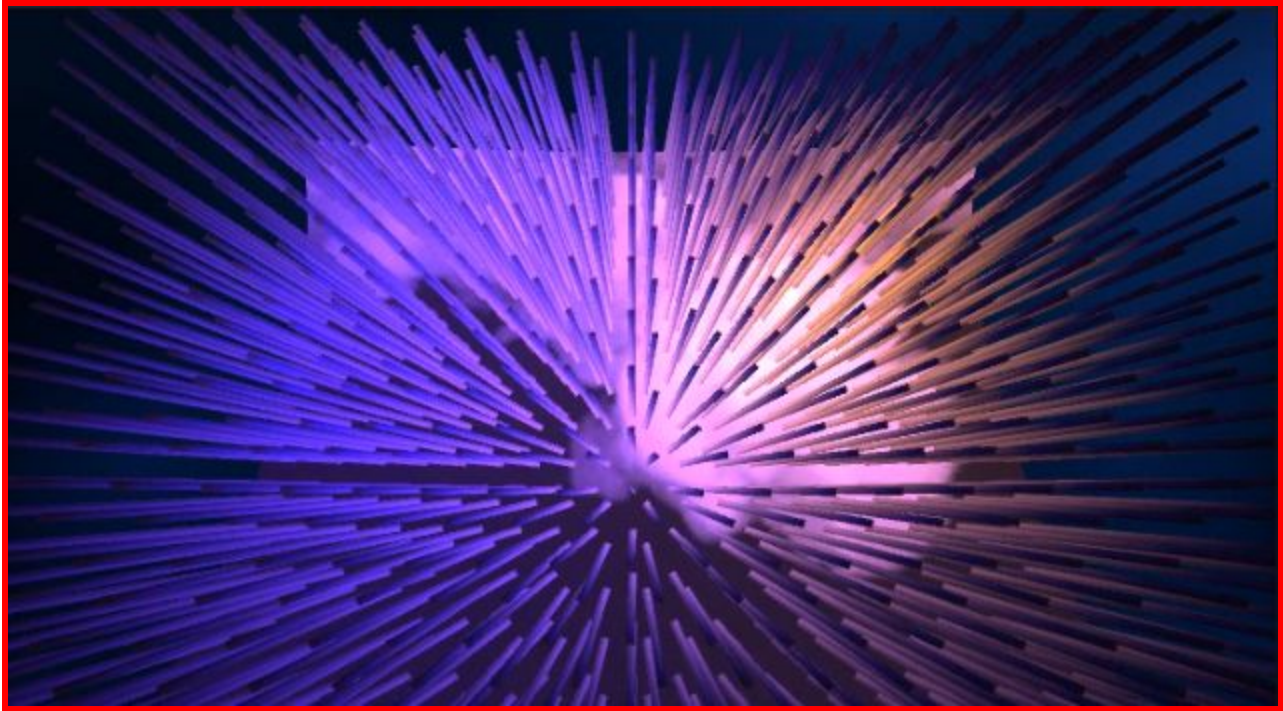


Depth of Images

2020. Fall: Advanced Immersive Media Programming

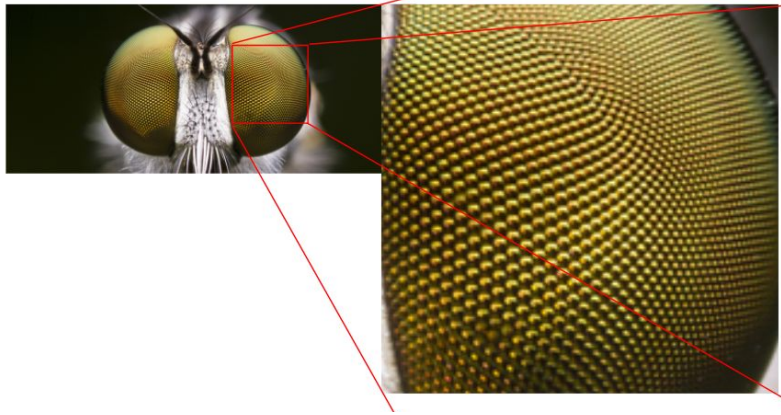
320200098 . 석 혜 정 (Hae Jung Suk) <https://github.com/dbdip/AIMPFinal>



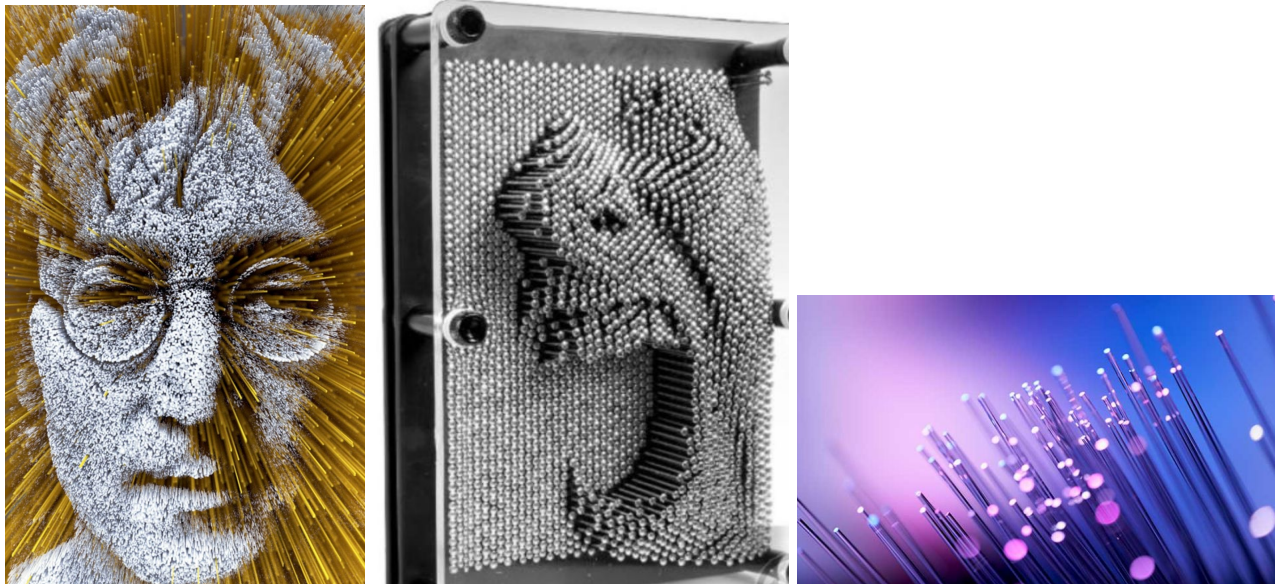
Motivation

1. Insect's compound eye

- 곤충의 겹눈은 수백개에서 수만개(잠자리)의 카메라 렌즈를 촘촘하게 연결하여 구형(Sphere)을 이루는 구조를 갖는다.
- 이러한 구조는 넓은 시야(large field of view), 낮은 수차(low aberrations), 무한한 피사계 심도(infinite depth of field) 등 많은 장점을 갖는다[].
- 본 프로젝트를 위해 곤충 겹눈의 구조와 그 장점으로 부터 영감을 얻어 작품의 규모와 왜곡, 깊이의 표현을 선택적으로 상징화 하였다.



2. Pull, Shape, Light, Interaction, VR



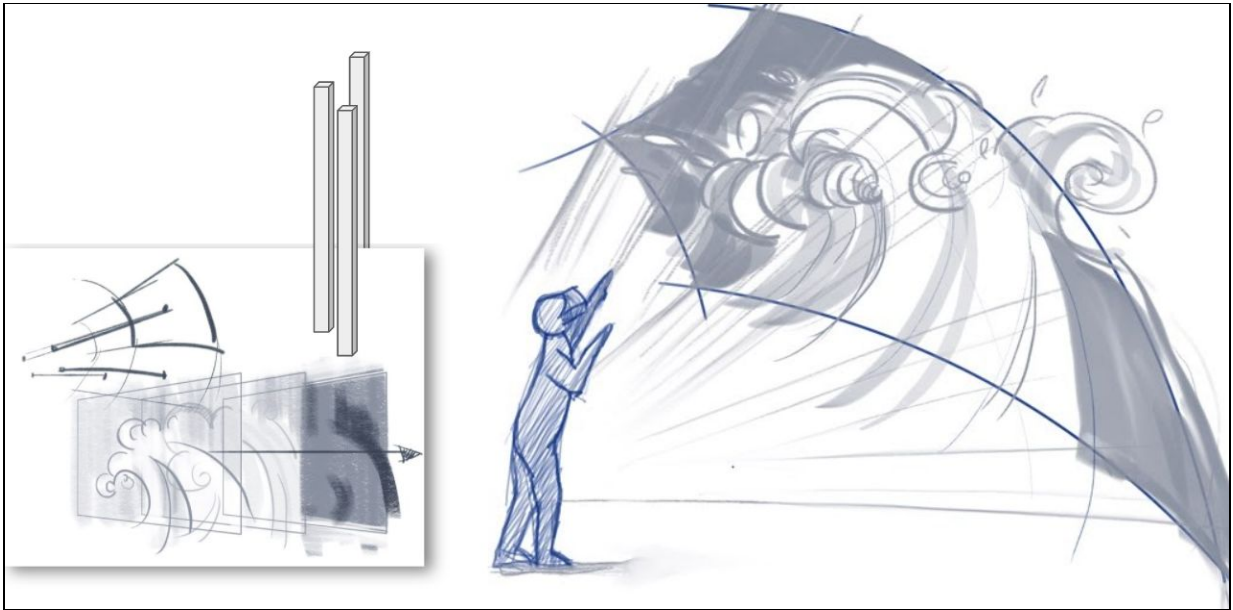
- 깊이를 갖는 구조물 중 수천개의 핀으로 형상을 만드는 핀아트(Pin art)를 레퍼런스로 삼았다.
- 선은 광섬유(electronic fiber)와 같이 주변의 빛을 반사하며 그 자체로 재질적 특징을 갖지 않도록 한다.
- 실사영상에서 깊이채널을 이용하여 카메라와 가까운 부분은 짧게 카메라에서 멀리 떨어져 있는 부분은 길게 뻗어나오는 광섬유 핀아트 구조물을 가상환경에서 감상할 수 있도록 제작한다.

Dynamic Animation in Virtual Environment

1. Concept drawing

- 가상현실에서 거대한 핀 구조물이ダイナミック한 이미지시퀀스에 따라 애니메이션 되는

장면을 기본 컨셉으로 삼았다.



2. Frameworks for the pin structure

- Step1 Machine Learning

2D 영상으로 부터 깊이(카메라와의 거리)를 추정(Depth prediction)하여 0-1의 value로 출력하는 알고리즘을 찾아 데이터를 학습시킨 후, Depth channel을 얻는다
Depth channel sequence를 이미지의 feature 상실을 최소화하여 데이터 크기를 줄인다.

- Step2 Scripting in Unity

Mesh의 삼각형 중심으로 부터 Normal Vector를 향한 선을 굵도록 스크립팅한다.
Mesh에 Step1에서 준비한 이미지시퀀스를 mapping하여 그 Value값을 받아 선의 길이가 주어지도록 스크립팅한다.

- Step3 Adding a sense of art for outcome

Custom shader로 빛과 주변을 반사하는 광섬유 재질을 만들고 라이팅을 한다.

Framework in Machine Learning

1. MiDas: Depth Estimate by Deep learning algorithm

- MiDaS는 다목적 최적화를 사용한 10 가지 개별 데이터 세트를

비지도학습(Unsupervised learning:CNN)하여 단일 이미지에서 상대적인 역 깊이를 계산한, 단안심도측정(Monocular Depth Estimation)이 가능하도록 설계된

알고리즘이다.

- Journal paper [Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer](#)
- Github code download <https://github.com/intel-isl/MiDaS>
- 최신 버전의 [PyTorch](#), [Torchvision](#), [OpenCV](#), [Visual C++ Redistributable](#) 패키지를 설치해서 사용한다.
- 프로젝트를 위한 최초의 테스트 버전은 [Anaconda navigator 3](#)의 [Jupyter Notebook](#)에서 [python 3.7](#)로 작성하여 제작하였다.



[test run python script](#)

2. Data creation

- Data preprocessing: 스크린에 디스플레이될 영상(Cloud_chamber.mov HD 1920x1080)은 movie파일로 SD format (640x480)로 리사이징 하여 사용하였다.



- 11분이상의 시퀀스가 3분 남짓으로 재생되도록 고속촬영된 movie파일로 0-2667 frame의 jpg 파일을 생성하여 학습시키기 보다는 머신러닝API를 활용하여 movie파일 자체로 부터 [depth estimated movie](#)파일을 얻었다.
- 문제점으로 MiDas로 학습시킨 결과 아래 아티클의 샘플 시퀀스와 마찬가지로 전반적인 flashing 현상이 있었다.

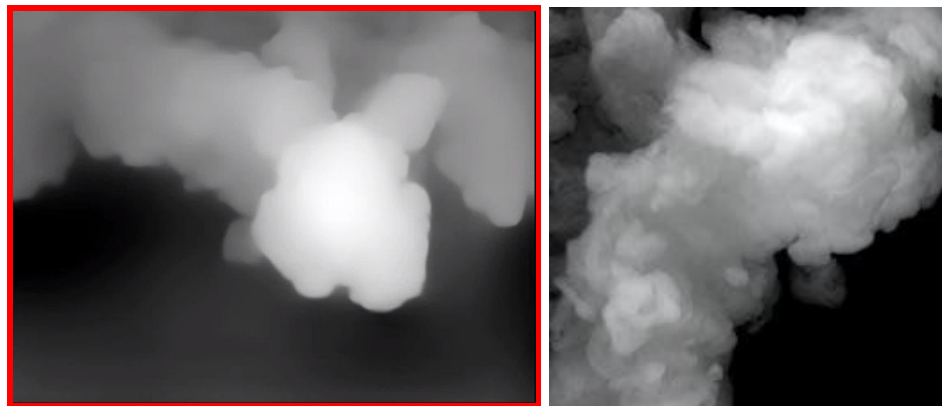
<https://heartbeat.fritz.ai/research-guide-for-depth-estimation-with-deep-learning>

[g-1a02a439b834](https://github.com/1a02a439b834)

- 문제해결방법으로 'FrameBlend' time node를 사용하여 적용시켜 문제가 되는 번쩍거림을 2D 영상소프트웨어 [NUKE](#)를 사용하였다. (앞뒤 프레임의 차이를 보간하는 방법으로)



- 깊이에측으로 얻은 결과(좌)와 오리지널 영상을 Multiply하여 표면의 텍스처링과 깊이가 동시에 반영되는 이미지 시퀀스(우)를 제작하였다.



3. Feature Reduction

- 실시간 렌더링 엔진에서 실행시키기 위해 Jupyter Notebook으로 python 스크립트를 작성하여 형상(feature) 손실을 최소화하여 224x224 사이즈의 이미지 시퀀스로 feature reduction을 하였다.

```
files_crop = [file_i
               for file_i in os.listdir('source/img_SM_224_224/')
               if file_i.endswith('.jpg')]

#cropped images load
imgs_crop = []
for file_i in files_crop:
    a = plt.imread(os.path.join('source/img_SM_224_224/', file_i))
    imgs_crop.append(a)

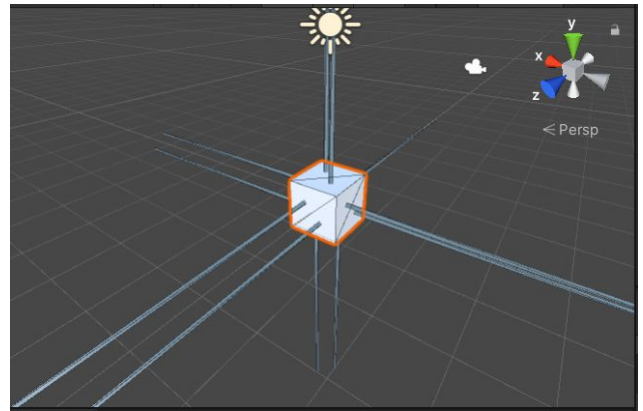
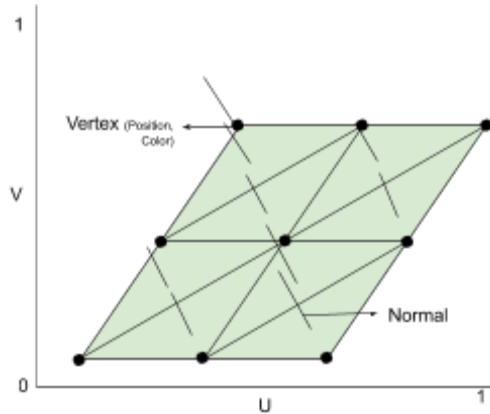
imgs_crop = np.array(imgs_crop)
imgs_crop = np.float32(imgs_crop)
```

Framework in Unity

1. Goals for the scripts

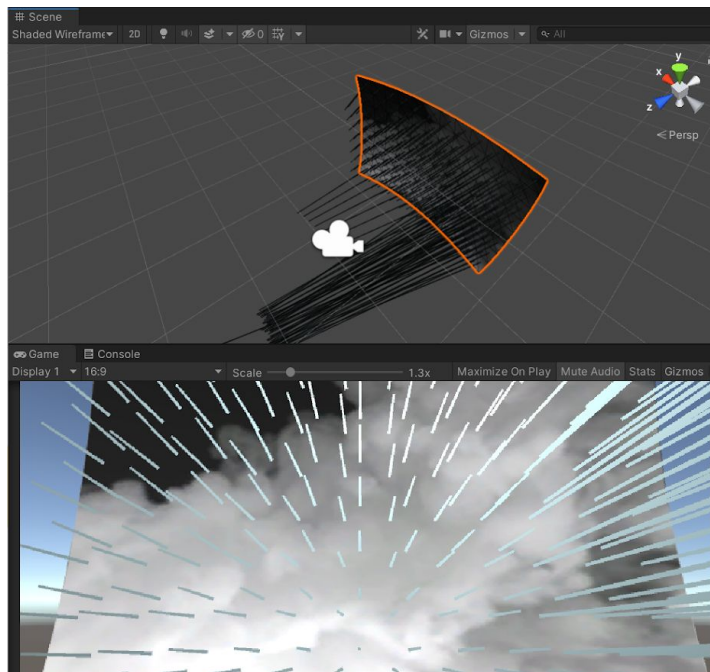
- Goal 1. Finding the start position and the direction of the lines
- Goal 2. Drawing thousands of lines

삼각형 메쉬의 정점(Vertex)를 이용해서 각 삼각형의 중심점 center를 찾고, center로부터 표면에 수직인 Normal 방향으로 수천개의 선을 그린다.



- Goal 3. Getting the value from the current texture image for deciding the length of the lines

Texture image의 value를 이용해서 서로 다른 길이의 선을 출력한다.



=> 정확하게 Value값을 반영하지 못함

- **Goal 4. Replacing the still image texture to the animated image sequence**

Texture image를 animated sequence mapping으로 전환하고 resources로 앞에서 얻은 Depth of image sequence로 바꾸기

2. C# Scripting

G1.Start and direction of lines	G2 Multiple line drawings	G3 Line length	G4 Texturing image sequence

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class AllScriptFinal : MonoBehaviour
{
    Mesh startMesh;
    int[] triangles;

    [HideInInspector]
    public Vector3[] vertices;

    public LineRenderer linePrefab;
    private float[] dist; //distance
    LineRenderer[] facelines;

    [HideInInspector]
    public Color[] colors;

    //An array of Objects that stores the results of the Resources.LoadAll() method
    private Object[] objects;
    //Each returned object is converted to a Texture and stored in this array
    Texture2D[] textures;
    //With this Material object, a reference to the game object Material can be stored
    private Material goMaterial;
    //An integer to advance frames
    private int frameCounter = 0;

    void Start()
    {
        //Get a reference to the Material of the game object this script is attached to
        this.goMaterial = this.GetComponent<Renderer>().material;

        //Load all textures found on the Sequence folder, that is placed inside the resources folder
```

```

this.objects = Resources.LoadAll("Sequence", typeof(Texture2D));

//Initialize the array of textures with the same size as the objects array
this.textures = new Texture2D[objects.Length];

//Cast each Object to Texture and store the result inside the Textures array
for (int i = 0; i < objects.Length; i++)
{
    this.textures[i] = (Texture2D)this.objects[i];
}

startMesh = this.GetComponent<MeshFilter>().mesh;
triangles = startMesh.triangles;
vertices = startMesh.vertices;

// build facelines
int i1, i2, i3;
Vector3 v1, v2, v3;
faceLines = new LineRenderer[triangles.Length / 3];
dist = new float[triangles.Length / 3];
for (int i = 0; i < triangles.Length / 3; i++)
{
    i1 = triangles[(i * 3) + 0];
    i2 = triangles[(i * 3) + 1];
    i3 = triangles[(i * 3) + 2];
    v1 = vertices[i1];
    v2 = vertices[i2];
    v3 = vertices[i3];

    // Debug.Log(message: $"vertices of {i}th triangle = {triangles[i * 3 + 0]}, {triangles[i * 3 + 1]}, {triangles[i * 3 + 2]}\\n");

    Vector3 center = (v1 + v2 + v3) / 3;

    // Debug.Log("center position is " + center);
    // Debug.Log("end Position is " + (center + normal * dist[i]));

    LineRenderer line = Instantiate(linePrefab, center, Quaternion.identity);
    line.SetPosition(0, center);
    line.SetWidth(0.03f, 0.015f);

    faceLines[i] = line;
}
}

void Update()

```



```

{
    //Call the 'PlayLoop' method as a coroutine with a 0.04 delay
    StartCoroutine("PlayLoop", 0.022f);

    //Set the material's texture to the current value of the frameCounter variable
    this.goMaterial.mainTexture = textures[frameCounter];

    LinesFromCenterOfTriangle(triangles, vertices);
}

//The following methods return a IEnumerator so they can be yielded:
//A method to play the animation in a loop
IEnumerator PlayLoop(float delay)
{
    //Wait for the time defined at the delay parameter
    yield return new WaitForSeconds(delay);

    //Advance one frame
    frameCounter = (++frameCounter) % textures.Length;

    //Stop this coroutine
    StopCoroutine("PlayLoop");
}

public void LinesFromCenterOfTriangle(int[] triangles, Vector3[] vertices)
{
    Vector3 center;
    Vector3 normal;

    Debug.Log($"triangles length = {triangles.Length}");
    //Debug.Log($"vertices length = {vertices.Length}");

    float H, S, V;
    int i1, i2, i3;
    Vector3 v1, v2, v3;

    for (int i = 0; i < triangles.Length / 3; i++)
    {
        i1 = triangles[(i * 3) + 0];
        i2 = triangles[(i * 3) + 1];
        i3 = triangles[(i * 3) + 2];
        v1 = vertices[i1];
        v2 = vertices[i2];
        v3 = vertices[i3];
        center = (v1 + v2 + v3) / 3;
        normal = GetNormalVectorToTriangle(v1, v2, v3);
    }
}

```

```

// use barycentric coordinates to interpolate the mesh
// location to a texture location
Vector2 uvA = startMesh.uv[triangles[(i * 3) + 0]];
Vector2 uvB = startMesh.uv[triangles[(i * 3) + 1]];
Vector2 uvC = startMesh.uv[triangles[(i * 3) + 2]];

float a, b, c = 0;
Barycentric(center, v1, v2, v3, out a, out b, out c);

```

//follow the texture coordinate mapping process, [barycentric interpolation cs script](#)

```

Vector2 uvP = a * uvA + b * uvB + c * uvC;
int x = Mathf.FloorToInt(uvP.x * textures[index].width);
int y = Mathf.FloorToInt(uvP.y * textures[index].height);

// get color of the texture at the position that matches
// the center of the mesh triangle
Color color = textures[frameCounter].GetPixel(x, y);

Color.RGBToHSV(color, out H, out S, out V);
// Debug.Log("V: " + V);
//dist[i] = Mathf.Gamma(V, 5, 1/1.5f) + 0.5f;
dist[i] = V * 2.5f;

// update the line's end position
Vector3 endPosition = center + (normal * dist[i]);
faceLines[i].SetPosition(1, endPosition);
}
}

```

```

void Barycentric(Vector3 P, Vector3 A, Vector3 B, Vector3 C, out float a, out float b, out float c)

```

```

{
    Vector3 v0 = B - A, v1 = C - A, v2 = P - A;
    float den = v0.x * v1.y - v1.x * v0.y;
    b = (v2.x * v1.y - v1.x * v2.y) / den;
    c = (v0.x * v2.y - v2.x * v0.y) / den;
    a = 1.0f - b - c;
}

```

```

public Vector3 GetNormalVectorToTriangle(Vector3 v1, Vector3 v2, Vector3 v3)
{
    Vector3 v1v2 = v2 - v1;
    Vector3 v2v3 = v3 - v2;
    Vector3 normal = Vector3.Cross(v1v2, v2v3);
}

```

```
    return normal;
  }
}
```

- G1. Finding start position and direction of lines
 - LinesFromCenterOfTriangle() : 삼각형 메쉬의 중심점을 찾는 함수를 정의
 - GetNormalVectorToTriangle() : 삼각형의 법선 벡터를 구하는 함수를 정의
- G2. Drawing multiple lines
 - Instantiate() : G1에서 구한 start point와 방향으로 여러개의 lineRenderer 생성
- G3. Getting the value of line length from the animated image textures
 - Barycentric() : 삼각형의 세 정점을 이용해 삼각형의 무게중심점 uvP의 좌표를 구하기 위해 Barycentric Interpolation 공식을 사용(<https://wiki.unity3d.com/index.php/Barycentric#Barycentric.cs>)¹
 - GetPixel() : 삼각형 중심점의 uvP에 대응되는 texture x, y를 찾고 이에 해당하는 texture image pixel의 R, G, B 값 획득
 - RGBToHSV() : R, G, B값을 H, S, V로 변환하여 밝기에 해당하는 Value값을 라인의 길이 array인 dist로하는 endPoint를 설정
- G4. Texturing image sequence
 - Resources.LoadAll() : Resources 디렉토리 안의 "Sequence" 폴더를 찾아 어셋을 모두 load.
 - StartCoroutine() : yield, IEnumerator()와 함께 frameCount의 play time을 조절

3. Line Shading

- Line texturing에 다음과 같은 맵을 사용해서 다소 flat해 보일 수 있는 Line렌더러에 입체감을 주고, 라인 끝부분의 경계를 구분할 수 있도록 하였다.

¹ 이 부분에 대해 도움을 받고 설명을 들었으나 공식에 대해 완벽하게 이해하지 못했습니다.



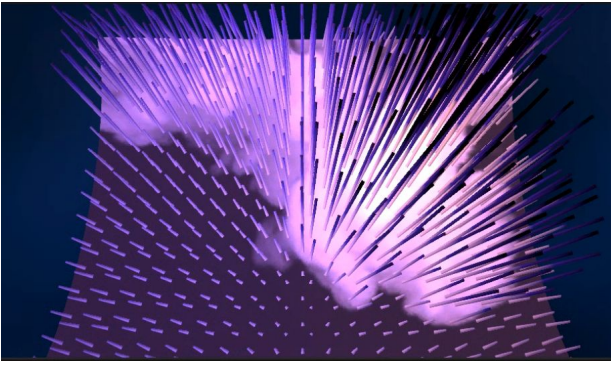
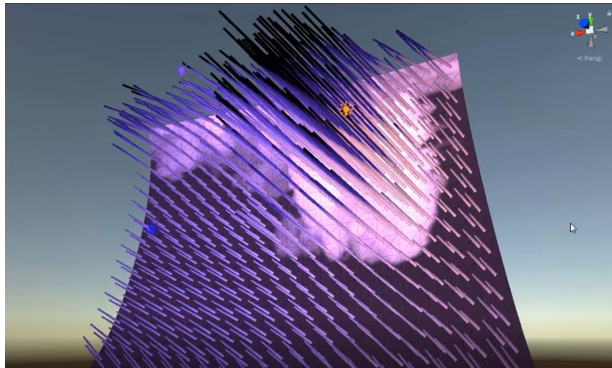
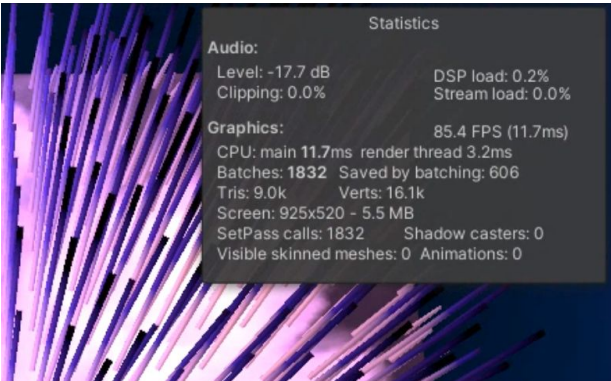
- Cubemap shader 적용

```
Shader "Custom/ColdMetal"
{
    Properties{
        _MainTex("Texture", 2D) = "white" {}
        _Cube("Cubemap", CUBE) = "" {}
    }

    SubShader{
        Tags { "RenderType" = "Opaque" }
        CGPROGRAM
        #pragma surface surf Lambert
        struct Input {
            float2 uv_MainTex;
            float3 worldRefl;
        };
        sampler2D _MainTex;
        samplerCUBE _Cube;
        void surf(Input IN, inout SurfaceOutput o) {
            o.Albedo = tex2D(_MainTex, IN.uv_MainTex).rgb * 0.5;
            o.Emission = texCUBE(_Cube, IN.worldRefl).rgb;
        }
        ENDCG
    }

    Fallback "Diffuse"
}
```

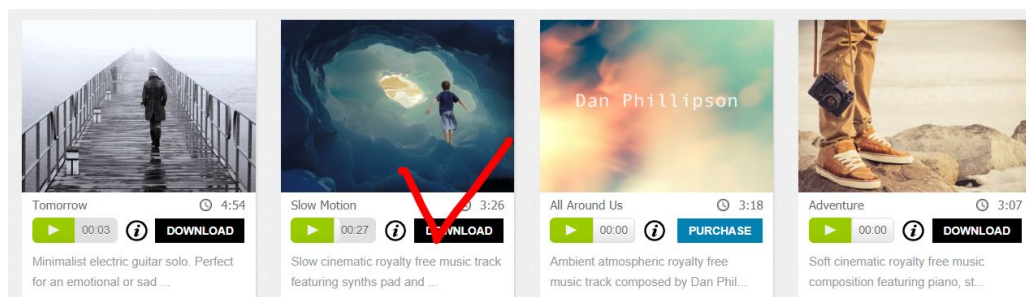
4. Final Output

	
<ul style="list-style-type: none"> - Final output (from Camera) https://drive.google.com/file/d/1a-F8pIYrNRWdPPAKvBN6ulr9T8sAtvdX/view?usp=sharing 	<ul style="list-style-type: none"> - Final output (from Perspective Camera) https://drive.google.com/file/d/1hdHH3QKBuKtAPkuaOs-C9C7APWE7iCly/view?usp=sharing
 <p>Statistics</p> <p>Audio: Level: -17.7 dB DSP load: 0.2% Clipping: 0.0% Stream load: 0.0%</p> <p>Graphics: 85.4 FPS (11.7ms) CPU: main 11.7ms render thread 3.2ms Batches: 1832 Saved by batching: 606 Tris: 9.0k Verts: 16.1k Screen: 925x520 - 5.5 MB SetPass calls: 1832 Shadow casters: 0 Visible skinned meshes: 0 Animations: 0</p>	<ul style="list-style-type: none"> - Performance Hardware Specification:

Others

1. Sound Track

- 음악은 www.bensound.com (Royalty free music) 에서 구매/ 다운로드 하여 사용하였다.
- Slow Motion 3:26



Epilogue

이 프로젝트는 2020년 2학기 Special Topics on Creative Technologies와 Advanced Media Immersive Programing 수업의 학기말 과제로 제작되었다. 전반적으로는 기술 목표로 잡았던 모든 단계를 구현했다고 볼 수 있지만 현재(2020-12-24)까지는 360도 영상은 촬영 시퀀스를 사용하여 파노라믹 뷰에서 뿔어나오는 라이팅 퍼포먼스를 적용하지 못하였다. 머신러닝에 대한 기본지식 부족과 프로그래밍의 미숙함으로 기획부터 고급 프로그래밍을 피할 수 있도록 하였으나 여전히 작업과정은 쉽지 않았다. 결국 그래픽 결과물의 낮은 완성도를 조금이라도 끌어올리기 위해 여러가지 부수적인 작업(2D영상 소프트웨어와 오디오, 결과물의 편집 등)을 거치게 되어 아쉬움이 남는다. 주요 피드백으로는 대형의 실물 설치가 가능할 것과 인터랙션, 그리고 머신러닝으로 학습시킨 오디오에 따라 변화하는 핀아트를 고려해 보는게 어떨겠냐 는 피드백을 받았는데 분야에 대한 이해와 지식을 넓혀 향후에 다양한 시도 도전해볼 수 있을 듯 하다. 첫 머신러닝과 유니티 코딩 과제를 어느 정도 수행 하여 결과를 제출할 수 있게 되어 감사하게 생각한다.