February 17, 2025

Chapter 1

Build

1.1 CMake

Basic notions of CMake: target(executable, library), link

We can install cmake with brew.

1.2 Libraries

CMake, glfw, glew, glm, `stb_image`, gltf

1.2.1 GLEW vs GLAD

They are extension loading library which allows us to use the later version of OpenGL. They integrate some problems possibly caused by different platforms. GLAD and GLEW are the most popular two choices. We need to include it before GLFW.

GLAD - 웹페이지에서 다운받아 설치 없이 직접 넣는다. - 모든 소스파일에 include해주어야 한다. (아마)

GLEW 주의점 - glew는 brew로 설치한다. - inlay hint가 보인다. - 초기화할 때 glfw창 만든 후 glewInit ()해야 한다.

1.2.2 GLFW vs GLUT

창, 컨텍스트, surface(?), 인풋, 이벤트 핸들링 을 한다

GLFW 특징 - brew로 관리가능하다.

1.2.3 glm

brew로 관리가능하다.

1.2.4 stb image

brew로 관리가능하다.

1.2.5 imgui

git에서 다운받아 설치 없이 직접 넣는다.

Chapter 2

Windows

2.1 What is OpenGL?

2.1.1 Graphics API

OpenGL is a software interface to graphic hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

우리가 만들고 싶은 어플리케이션을 하나 다른 소프트웨어나

그래픽스 API를 우리의 어플리케이션이 화면에 그림을 그리기 위해서 그래픽 카드와 소통하기 위한 규약이라고 정의내려 볼수도 있을 것 같습니다. 어플리케이션 내부에서 그래픽 카드에 삼각형의 위치 정보, 색깔 정보 등을 보내서 계산시키게 한 다음 각 픽셀마다 출력해야 하는 색상 값을 그래픽 카드로부터 받아서 모니터에 전달하여 출력하게끔 하는 그런 함수들의 기능설명서에 비유해 보아도 좋을 것 같습니다.

API는 정의 및 프로토콜 집합을 사용하여 두 소프트웨어 구성 요소가 서로 통신할 수 있게 하는 메커니즘입니다. 예를 들어, 기상청의 소프트웨어 시스템에는 일일 기상 데이터가 들어 있습니다. 휴대폰의 날씨 앱은 API를 통해 이 시스템과 "대화"하고 휴대폰에 매일 최신 날씨 정보를 표시합니다. API는 Application Programming Interface(애플리케이션 프로그램 인터페이스)의 줄임말입니다. API의 맥락에서 애플리케이션이라는 단어는 고유한 기능을 가진 모든 소프트웨어를 나타냅니다. 인터페이스는 두 애플리케이션 간의 서비스 계약이라고 할 수 있습니다. 이 계약은 요청과 응답을 사용하여 두 애플리케이션이 서로 통신하는 방법을 정의합니다. API 문서에는 개발자가 이러한 요청과 응답을 구성하는 방법에 대한 정보가 들어 있습니다.

- OpenGL 크로노스 그룹 크로스 랭귀지, 크로스 플랫폼 - DirectX - Vulkan - Metal

GPGPU API: CUDA, OpenCL

윈도우 관리 못함, 인풋 관리 못함, gui 없음

2.1.2 Implementatations

엔비디아에서 만든 C:/Windows/System32/NvIFROpenGL.dll 구현체

2.2 OpenGL object

2.3 OpenGL context

컨텍스트가 만들어지면 default framebuffer가 하나 바로 만들어진다.

한 컨텍스트는 응용프로그램의 창 같은 viewable surface 하나를 표현할 수 있다. 한 컨텍스트는 여러 OpenGL 오브젝트들을 갖는다. 대부분 다른 컨텍스트와 공유 가능하지만, 싱크, GLSL, 컨테이너, 쿼리 오브젝트는 공유불가능.

커렌트 상태가 되어야 한다. 바인드같은 느낌인 듯 모든 OpenGL 커맨드는 커렌트 상태의 컨텍스트에 영향을 미친다. 스레드마다 하나씩 있어서 한 프로세스가 여러 개의 커렌트 컨텍스트를 갖는 것도 가능은 하겠지만 싱글 스레딩인 경우는 안될 것.

2.4 Older OpenGL

2.4.1 Compatibility issue

옛날 opengl 기능들에 접근할 수 있는 여러 방법들이 시도되다가 core context와 compatibility context로 나누는 방식이 정착함

2.4.2 OpenGL 1.x

SGI(실리콘 그래픽스)사의 그래픽 라이브러리 GL의 오픈소스화, 1992년 첫 발표

fixed function pipeline(standard pipeline, immediate mode) ‘glBegin‘ ‘glEnd‘ 같은 것

2.4.3 OpenGL 2.x

2.0과 2.1만 존재 programmable pipeline with shaders vertex shader fragment shader

2.4.4 OpenGL 3.0

OpenGL 3.0 - deprecation model이 도입됨 - 여기서부터 compatibility가 깨지게 될 줄 알았으나 full 3.0 context 기준으로 deprecated 표기만 있었을 뿐임 - fixed function pipeline(immediate mode)도 deprecated functionality에 포함됨.

2.4.5 OpenGL 3.1

OpenGL 3.1 - 3.0에서 deprecate된 기능 remove - wide line에 관한 예외 하나 있어서 remove되지 않음 - 3.0에서 deprecate된 것 말고 추가 deprecate 없음

대신 `ARB_compatibility`라는 새로운 익스텐션이 도입되어서 이걸 사용하면 removed feature를 사용할 수 있었으므로 완전히 대체된 건 또 아니었음

2.4.6 OpenGL 3.2

OpenGL 3.2 core profile - 3.1과 똑같이 deprecate된 기능 remove - 3.1 상위호환 - 3.1에서 remove 없는 deprecate 하나 추가

OpenGL 3.2 compatibility profile - 3.1의 `ARB_compatibility` 상위호환

Core profile과 Compatibility profile이라는 개념이 도입되었다. 둘 중 하나를 반드시 선택하여 context를 생성해야 함. 즉, compatibility profile의 구현을 의무화시킴으로써 extension이 아닌 3.2의 정식 스펙으로 받아들인 것.

2.4.7 OpenGL 3.3

3.3 core profile - 3.2 core profile 상위호환 3.3 compatibility profile - 3.2 compatibility profile 상위호환

왜 3.3이 first modern OpenGL로 불리는가? - 3.3은 사실 3.2와의 차이는 거의 없는데 같은 날 발표된 4.0과는 차이가 큼. 그래서 profile 개념을 사용하고 가장 가벼우면서 현재까지 compatible한 3.2와 3.3 중 그나마 최신의 3.3을 사용한다는 느낌인 것 같음. - GLSL 버전이 1.5에서 3.3으로 뛰어서 그럴 수도 있을 것 같음. 근데 GLSL 1.5와 3.3도 큰 차이가 생기지 않았음. - 게다가 사용하는 입장이 아니라 그래픽카드에 OpenGL을 구현해야 하는 개발자 입장에서는 3.3이 훨씬 어려워서 버그도 많다고 함. (AMD가 구현한 게 대표적인 예라는 듯) - 그냥 다 필요없고 애플이 Metal 발표 전 OpenGL 3.3과 4.1을 표준버전으로 지원했기 때문에 다들 우루루 따라간 게 아닌가 하는 생각이 사실 ㅈㄹ 강하게 듦. - 몇몇 튜토리얼(learnopengl 포함)에서 현재까지 발표된 버전 및 미래 버전에서 호환되는 것을 보장하기 때문에 3.3을 사용한다는 표현들을 찾을 수 있는데, 호환성 자체는 틀린 말은 아니지만 사실 3.1 이후 모든 버전에도 해당되는 말이기 때문에 마치 3.1과 3.2는 호환성이 없다는 오해를 불러일으키기 좋은 듯함. - 3.3부터 D3D11와의 관계에도 뭔가 있다는 얘기도 본 것 같은데 이거까지는 알아보지 못했음.

너무 모르겠음.

2.4.8 OpenGL 4.0

compute shader

2.4.9 Forward compatibility

OpenGL로 상위호환 개념을 처음 접하면 무진장 헷갈리는 개념.

forward(upward) compatibility(상위호환) - backward compat의 반대 - 위키피디아(영어)에 찾아보면 잘 나와있음

forward compatibility bit set - opengl에서는 기존 의미로부터 변형되어 context의 속성으로서 그냥 deprecated된 기능들을 remove해서 더 이상 지원하지 않는다는 의미로 쓰이는 것 같음. 왜 그렇게 됐는지는 모르겠음. - 재미있게도 조금의 혼동을 피하기 위함인지 공식스펙 문서에서는 위의 상위호환을 뜻하는 단어로 upward compatibility를 사용함. - 3.0부터 존재하는 개념이라 2.1로 컴파일 해보면 다음 에러를 확인할 수 있음: 'Error 65540: Forward-compatibility is only defined for OpenGL version 3.0 and above' - 맥에서는 core profile을 사용할 때 반드시 forward compatibility bit set을 사용하게끔 강제되어 있는데 이유는 모름.

2.5 Variations

2.5.1 WGL

2.5.2 OpenGL ES

2.6 References

-https://www.khronos.org/opengl/wiki/Main_Page -https://registry.khronos.org/OpenGL/index_gl.php -https://community.khronos.org/t/opengl-3-2-vs-3-3/63951 -https://support.apple.com/en-us/HT202823

Unified Particle Physics for Real-Time Applications

Chapter 3

Inputs

3.1 Initialization

‘glfwInit‘ 보통 if 조건문으로 넣고 에러 핸들링 ‘glfwTerminate‘ modern system에서는 리소스 freeing 자동으로 잘하지만 global system을 변경해야 할 땐 꼭 해줘야 한다. ‘glfwInitHint‘ glfwInit 전에 해야 하며 glfw 돌아가는 중간에 절대 수정되지 않는다. platform-specific 힌트도 존재한다. 세 개밖에 없다.

‘glfwGetError‘ ‘glfwSetErrorCallback‘

virtual screen과 content area

다른 스레드에서 호출될 수도 있는 함수들

3.1.1 Window

윈도우가 없는 컨텍스트도 있지만 우리가 사용할 사실상 모든 컨텍스트는 윈도우와 똑같은 개념이라고 생각하면 편하다.

‘glfwCreateWindow(800,600,”Title”,NULL,NULL)‘ ‘glfwSetWindowSize‘

지정한 컨텍스트만 없애려면 glfwTerminate() 말고 glfwDestroyWindow(window) 사용 그러면 더 이상 어떤 이벤트도 윈도우로 전달 안됨

‘glfwWindowHint‘

윈도우 사이즈, 프레임버퍼 사이즈, 컨텐트 스케일

3.1.2 Monitor

3.1.3 GLAD initialization

스레드에 커렌트 상태의 컨텍스트가 있어야 함

3.2 Input handling

버퍼 스왑 이후 이벤트 프로세싱 ‘glfwPollEvents‘ ‘glfwWaitEvents‘ 인풋받고 윈도우 안의 내용만 변경할 때 cpu 사이클 ‘glfwWaitEventsTimeout‘

3.2.1 Keyboard

key events, character events

3.2.2 Mouse

3.2.3 User pointers

3.3 Render loop

```
glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
```

3.3.1 Time

speed and sensitivity

3.3.2 Cleanup functions..?

더블 버퍼링 glfwSwapBuffers(window) every operation is done on the back buffer, to prevent flickering
프론트 버퍼와 백 버퍼가 있다.

glfwPollEvents() 아직 처리되지 않은 대기중인 이벤트 체크 보통 버퍼 스와핑 이후 처리 키보드 입력, 마우스 입력, 윈도우 크기 조정 glfwWaitEvents()?

3.3.3 Drawing commands

drawing commands

3.3.4 Render rate

deltatime

3.4 GUI

설정할 수 있는 인스펙터?를 만들어야 함 오브젝트들 설정 스카이박스 설정 카메라 설정 조명들 설정
셰이더 로더 모델 로더 (텍스쳐 로더? 모델 로더에 포함되나?) 만들어야 함

Chapter 4

Shaders

4.1 Standard pipeline

셰이더 없이 짜는 옛날 opengl

창 만들기와 렌더 루프 사이에 버텍스, 텍스쳐, 셰이더를 준비

- Vertex processing (눈으로 인식하기) - Vertex post-processing - Primitive assembly (스케치하기)
- Rasterization and fragment shading (색칠하기) - Per-sampling processing

transform feedback? asynchronous pixel transfers?

4.2 Shader stages

OpenGL의 렌더링 파이프라인은 다음 다섯 가지의 셰이더 stage를 정의한다. 스테이지가 실행되는 것을 invocation이라고 부르며, 매우 특수한 경우를 제외하고 각각의 invocation들은 상호작용할 수 없다.

- Vertex shaders - Tesselation shaders - control shader와 evaluation shader - Geometry shaders - Layered rendering, Transform feedback - Fragment shaders - Compute shaders

Vertex attributes, uniforms, textures

다섯가지 타입(fubid)과 두가지 컨테이너(vec,mat) rgba, stpq

버텍스 셰이더와 프래그먼트 셰이더 확장자 상관없다 Q. 적어도 초기 튜토리얼에서는 vert와 frag 사이에 아무것도 하지 않는데, 그렇다면 셰이더 파일 하나만 있어도 되는 건가? Q. 셰이더 프로그램의 최종결과물은 rgba를 나타내는 vec4이기만 하면 되는 건가?

4.2.1 Shader programs

드로잉 커맨드가 실행되면 바인드 되어 있는 program object와 pipeline object가 사용된다

4.3 Compilation

4.3.1 GLSL objects

GLSL objects do not follow the paradigm of OpenGL objects

Program object Shader object Program pipeline object

런타임 때 컴파일: 셰이더 코드와 텍스처 이미지 로드할 때 경로를 조심해야 한다. 셰이더 코드를 컴파일할 때 워킹 디렉토리는 오브젝트 파일이 존재하는 곳이다..? 예를 들어 clion에서 상대경로 지정할 때 실행파일이 서브디렉토리에 생기므로..? 불확실

셰이더 로더 클래스 만들기 1. 코드 읽기 최종적으로는 const char* 타입으로 변환시킬 것 2. 컴파일하기 셰이더 만들고, 소스 주고, 컴파일 프로그램 만들고, 셰이더 어태치하고, 링크 3. 유니폼 유틸리티 함수들 타입별로 버텍스 어트리뷰트는 안됨

4.4 Vertex shaders

input vertex stream에서 single vertex 받아서 계산 후 output vertex stream의 single vertex를 반환 모든 버텍스들이 버텍스 셰이더를 통과하는 것 보통 프로젝션 이전까지의 변환을 책임

대충 생각해서 버텍스 하나당 버텍스 셰이더가 대략 한 번씩 호출된다 생각하면 됨

정점 셰이더 layout (location = 0) VAO에서 받아오는 데이터들 `gl_Position` 미리 정의된 uniform 느낌? 진짜 uniform인진 모르겠다

4.4.1 Vertex stream

aPos aColor aTexCoord

4.4.2 VBO

4.4.3 VAO

셰이더에 정점 데이터 넣기 전에 먹기 좋게 자르는 작업 수행 각 인덱스마다

```
glVertexAttribPointer(index, size, type, normalized, stride, offset)
```

offset부터 size만큼의 데이터를 하나의 input으로 만든다 stride씩 뛰면서 input들의 array를 만든다 index(=location)는 그냥 표지이다

- glDrawArrays VBO vertex, normal, color -> primitives - glDrawElements EBO

```
glBindBuffer(GL_ARRAY_BUFFER, 0);
//glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
glBindVertexArray(0); // no need to unbind
```

$f(x)$

첫번째 줄 가능; VAO has stored the address of VBO 두번째 줄 불가능; VAO stores this unbind-call for EBO

4.5 Fragment shaders

인덱스로 그릴 때 프리미티브마다 인터폴레이션이 들어가므로 사각형을 어떤 두 삼각형으로 쪼개냐에 따라 그라데이션이 다르게 칠해짐

4.6 Shader class 만들기

기능 - 소스파일을 컴파일 - 유니폼변수 대입 - 특히 나중에 카메라 배울 때 만들 행렬들 -

Chapter 5

Meshes

배경

물체 context 만들어진 창, bind된 VAO, use선언된 셰이더 이 세 개가 주어지면 그릴 수 있는 듯?

한 모델이 여러 메시를 갖는다 포인터가 아니라 통째로 갖고 있다 모델을 그린다는 건 포함하는 메시를 전부 그린다는 것

Chapter 6

Materials

6.1 Wrapping

6.2 Filtering

Mipmap

loading, applying

직각삼각형이 아닌 primitive에 texture 입히면 찌그러지나?

Chapter 7

Cameras

7.0.1 Model matrix

키보드 신호나 물리엔진을 돌려 나온 위치벡터 등의 결과들로 Model 행렬을 만든다

7.0.2 View matrix

카메라의 위치 및 방향 정보로 View 행렬을 만든다

7.0.3 Projection matrix

절두체를 직육면체로 바꾼다 Projection 행렬은 다른 매트릭스에 비해서는 거의 안바뀌기 때문에 렌더 루프 바깥에서 잡는 경우가 많다 카메라의 줌인 줌아웃도 여기서 반영시킬 수 있다

Orthographic projection과 Perspective projection 두 가지 방식

perspective division - 이제 translation이나 rotation할 일 없으니까 homogeneous w component 버려도 됨 - x,y,z 성분 뽑고 z는 깊이니까 음 - 정점 셰이더 끝날 때 이루어진대. 최소한 이 예제에서는

즉 카메라 클래스가 view, projection 행렬을 만드는 메소드를 가져도 좋을 것 같다.

7.0.4 ViewPort transformation

7.1 9.4. Depth buffer

그냥 활성화시킨다는 것 말고 다루지 않음

구체적으로 각 행렬들이 glm에서 어떻게 만들어질 수 있는지들을 보자. glm documentation에 들어가면 stable extensions 에서 확인하는 것이 가능

```
GLM_EXT_matrix_transform
glm::lookAt(eye, center, up)
glm::rotate(mat, angle, axis)
glm::scale(mat, ratio_vec)
glm::translate(mat, vec)

GLM_EXT_matrix_projection

GLM_EXT_matrix_clip_space
```

MV matrix를 만드는 효과적인 방법들 - angles to axes - lookat to axes - rotation about arbitrary axis - quaternion - mat4 class?

10. Camera

뒤통수 컨벤션 Right(x) Up(y) Direction(z) 순서로 오른손

fov and aspects

lookAt(position, target, up) 은 view matrix를 만들어준다 - position vector 로 translation 먼저 함 Position을 (0,0,0)으로 보내는 행렬을 역행렬로서 구하게 되면

$$\begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- direction = position - target, right = up x direction 로 camera frame 구함 - Right를 (1,0,0), Up을 (0,1,0), Direction을 (0,0,1)로 보내는 행렬을 구하는데 이때 직교행렬의 역행렬이 전치행렬인 것 사용하면

$$\begin{bmatrix} R_x & U_x & D_x \\ R_y & U_y & D_y \\ R_z & U_z & D_z \end{bmatrix}^{-1} = \begin{bmatrix} R_x & R_y & R_z \\ U_x & U_y & U_z \\ D_x & D_y & D_z \end{bmatrix}$$

7.2 Euler angle

SO(3)를 위한 한 좌표계

RxRyRz roll->yaw->pitch?

Tait-Bryan angles, Aircraft principal axes

$D_x = \cos(yaw)\cos(pitch)$, $D_y = \sin(pitch)$, $D_z = \sin(yaw)\cos(pitch)$

yaw가 0이면 D=(1,0,0) yaw = -phi pitch = 90-theta

7.3 Three camera implementations

fly like camera FPS camera flight simulation camera

Chapter 8

Lighting