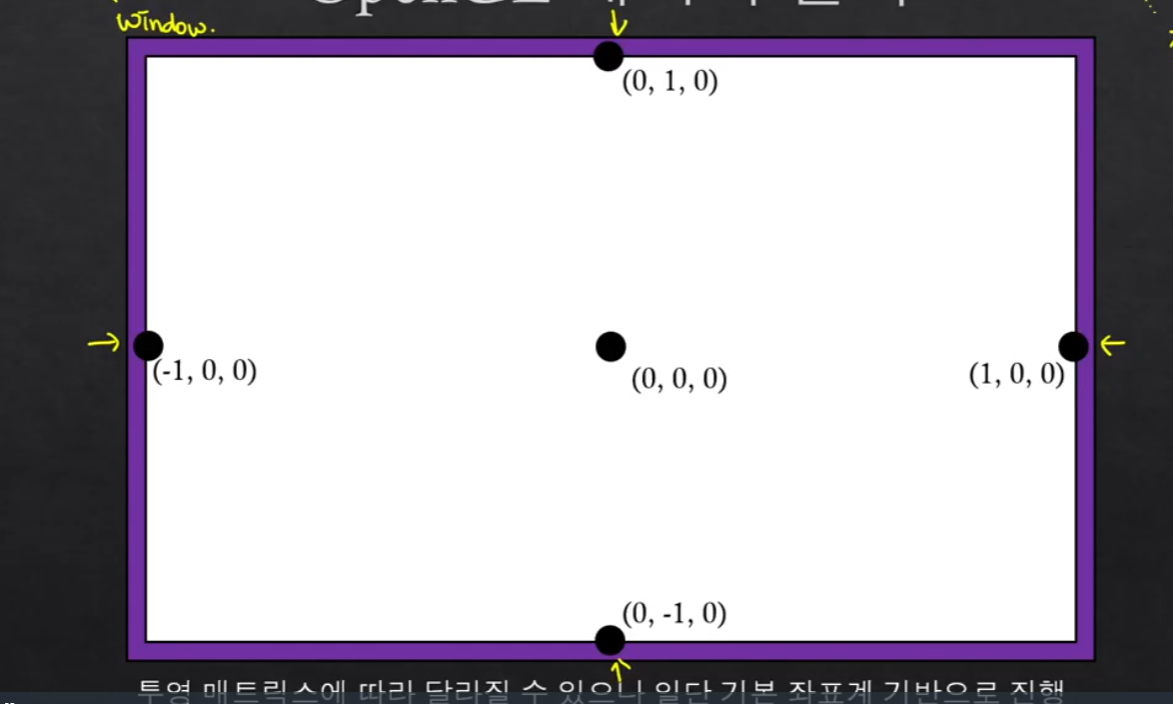
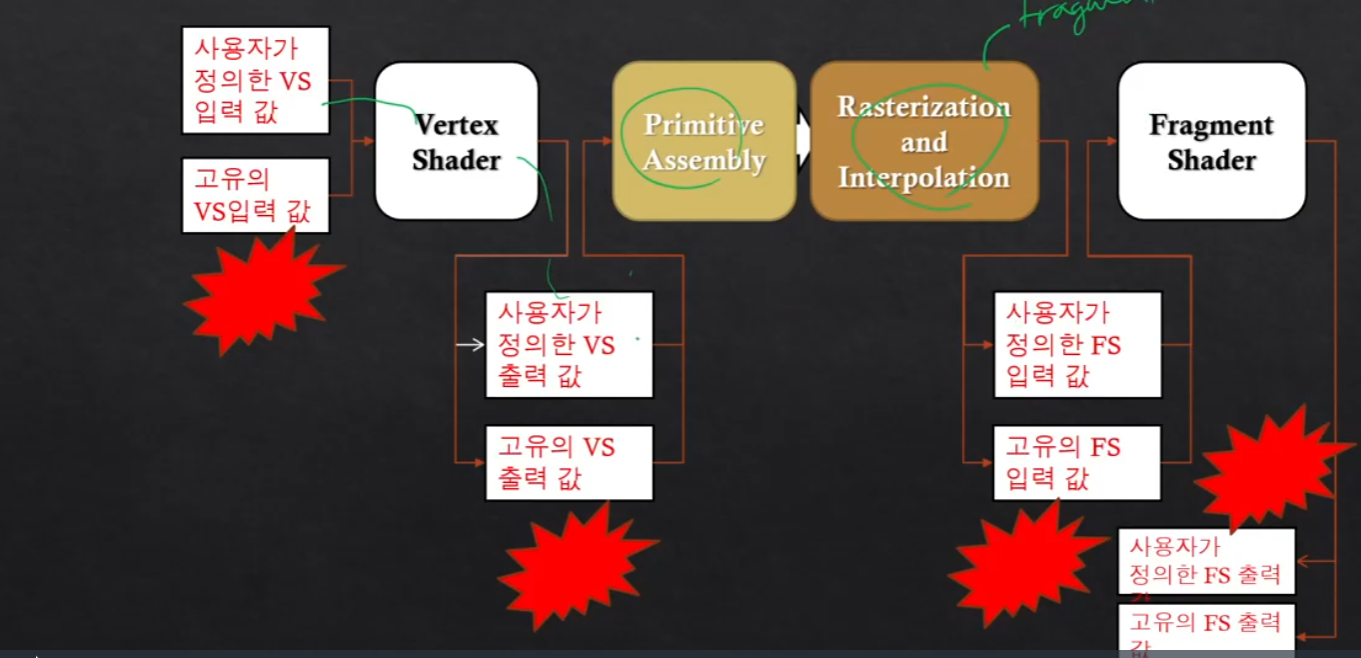
* 렌더링에 필요한 기본정보
* 점 (하나의 정점으로 이루어짐) ≠ vertex(정보의 묶음)
  + 삼차원 공간상에 정의되는 점
  + x,y,z값을 가짐
* 선(두개의 정점으로 이루어짐)
  + 삼차원 공간상에 정의되는 선
  + 최소 두 개의 점들로 이뤄짐
* 삼각형(세개의 정점으로 이뤄짐)
  + 삼차원 공간상에 정의되는 면
  + 세 개의 점들로 이뤄짐
* 점 -> Point  
  선 -> Line  
  삼각형 -> Triangle  
  정점 -> Vertex  
  요소 -> Primitive
* 좌표계 – 오른손
* 정육면체 - 36개의 정점이 필요(12개 삼각형)
* Graphics Pipline
  + 여러 단계에 걸쳐 진화함
  + 가장 큰 변화는 고정 그래픽스 파이프라인에서 프로그램 가능한 그래픽스 파이프라인으로 변화한것
* 고정 그래픽스 파이프라인
  + 텍스트이(가) 표시된 사진

    자동 생성된 설명
  + VT: 3D좌표 -> 2D좌표(투영 매트릭스)
  + PA: vertex->primitive
  + RI: primitive 내부를 채움, 점사이의 값을 보간함 (primitive -> Fragments(가공되기전 픽셀))
  + FO: 조명, 그림자 처리
  + FB: 모니터로 복사
* 프로그램 가능한 그래픽스 파이프라인
  + 텍스트, 전자기기이(가) 표시된 사진

    자동 생성된 설명
  + VS: 투영 + vertex animation (morphing)
  + FS: 다양한 필터링(라이트 셰프트, 블룸)
* Open GL 렌더링 구조
  + 일종의 State Machine 형태로 동작함
  + 파이프라인이 동작하기 전에(렌더링 시작전) 데이터는 이미 결정되어있다.(=Static) 파이프라인이 계속 퍼가는데 중간에 바뀌면 안된다!
  + 병렬화는 기본이다. 안되면 매우 비효율적(CPU나 GPU가 놀게 두면 안된다)
  + 효율적인 렌더링을 위해 고유의 Data 형식을 갖는다 -> Vertex Buffer Object(VBO)
  + 셰이더는 vertex 1개씩 처리된다.
* Open GL 데이터 준비
  + 
  + z 값은 Depth값으로 사용됨
  + face: 앞면, 뒷면 / 대부분 CCW
  + Vertex 데이터를 Array 형식으로 준비  
    float vertices[] 메인 메모리에 저장됨  
    GPU 메모리에 옮겨야한다
  + glGenBuffers(1,&VBO) //GLuint VBO  
    ID를 넣어줌
  + glBindBuffer(GL\_ARRAY\_BUFFER,VBO)  
    ID를 이용해 Bind함
  + glBufferData(GL\_ARRAY\_BUFFER,sizeof(vertices),vertices,GL\_STATIC\_DRAW)  
    데이터 할당
* Open GL 데이터 사용
  + 사용할 때 마다 Bind를 해라  
    BIND는 간단한 API라 자주해도 괜찮다  
    간혹 컨텍 스위치가 발생하는 것도 있으니 그 경우에는 조심
  + GL은 종류당(GL\_ARRAY\_BUFFER) 하나의 오브젝트만 BIND 허용
  + glVertexAttribPointer(0,3,GL\_FLOAT,GL\_FALSE,0,0)  
    Draw 시 데이터를 읽을 단위의 크기 및 시작점 설정
  + glDrawArrays(GL\_POINTS,0,1)// 몇번째 부터, vertex 갯수  
    primitive 선택, vertex갯수, 호출 시 즉시 GPU 동작
* Open GL Shading Language
  + Fragment Shader ≠ Pixel Shader
  + C 프로그래밍 형식
  + 집합체 타입(써줘야 효율이 좋다)
    - float: vec2, vec3, vec4, mat2. mat2x2~mat4,mat4x4
    - double: dvec2, dmat2 ~
    - int: ivec2 / uint uvec2 / bool bvec2
  + matrix는 Column major이다. 왼쪽부터 세로로 값이 순서대로 입력됨
  + x,y,z,w / r,g,b,a / s,t,p,q 사용가능
  + 텍스트이(가) 표시된 사진

    자동 생성된 설명
  + Access Modifier
    - in: 복사된 값을 받음
    - const in: 읽기 전용 복사 값을 받음
    - out, inout
    - 텍스트이(가) 표시된 사진

      자동 생성된 설명
* Shader 입출력
  + Shader 단위로 입력, 출력 값이 있다. 고유값과 사용자 정의 값이 있음
  + Vertex Shader: 최종 위치가 결정됨, 사용자 정의값이 파이프라인 외부에서
  + Fragment Shader: 사용자 정의 값 파이프라인 내부, 출력 값(외부)은 Deffered Shading에 사용되기도 함
  + 
* Vertex Shader 입력
  + 사용자가 정의한 VS 입력 값 = Attribute라고 칭함
  + glEnableVertexAttribArray(0) -> 0은 Attrib의 Location임, Attrib이 여러 개면 여러 번 해줘야함
  + layout(location = 0)in vec3 Position
  + layout 없이 사용가능
    - glGetAttribLocation(program, “Position”) // in vec3 Position
* Shader 외부 입력
  + 외부 입력 변수
  + shader당 주어지는 변수(Uniform)  
    시간
  + uniform 선언자
    - 외부에서 입력된 값, 셰이더 내부에서는 읽기만 가능, 쓰기는 불가능
    - 셰이더 전반에 걸쳐 읽기가 가능함
    - uniform float gScale
    - 이 친구도 ID,location 값을 사용함  
      glGetUniformLocation(program,”gScale”) 셰이더에서 사용하지 않으면 이상한 값이 리턴됨
    - glUniform{1234}{fdi ui}() //glUniform1f(ID,값)  
      v가 붙으면 array라는 뜻
    - glUniformMatrix
  + Vertex당 주어진 Data(Attrib)
* Shader 컴파일
  + shader program 생성  
    glCreateProgram()
  + shader object 생성  
    glCreateShader(ShaderType)
  + object에 source 할당 및 컴파일  
    glShaderSource(ShaderObj,1,p,Lengths)  
    glCompileShader(ShaderObj)
  + shader program attach  
    glAttachShader(ShaderProgram,ShaderObj)
  + Attach 완료 후 링크 수행  
    glLinkProgram(ShaderProgram)
  + 사용 시 함수를 불러줘야함  
    glUseProgram(program)
* Particle Effect
  + Quad 패치를 움직인다.
  + textur 기반 effect
* 프래그먼트 셰이더
  + 버텍스 셰이더의 출력과 프래그먼트 셰이더 입력의 타입은 서로 같아야 함
  + vertex -> primitive -> fragments 과정에서 따라서 보간이 일어남텍스트, 쇼지, 낱말맞추기게임이(가) 표시된 사진

    자동 생성된 설명
  + 버텍스는 두개이지만 보간된 프래그먼트는 10개
* Storage Qualifier
  + 각 stage 입출력을 지정해야함
  + Stage 자체의 입출력 값에 대해 쓰이는 Qualifier (비슷한 것 끼리)
    - in, attribute
    - out, varying
  + Access Modifier와는 다르다 (Stage 내부에서 쓰이는 Modifier)
  + 그래픽 파이프라인 외부입력 : 버텍스 셰이더 입력 값 (x,y,z입력),(r,g,b,a 입력)
  + 그래픽 파이프라인 이전 Stage로부터 이비력 : 버텍스 세이더의 출력값 == 프래그먼트 셰이더의 입력값
  + 내부에서 지정하고 변하지 않을 경우 const를 사용한다 각 셰이더 별로 선언해야함 ex) const float PI = 3.141592 버텍스와 프래그먼트 둘다 선언해야 둘다 사용가능
  + 버텍스 out -> 프래그먼트in == 버텍스 varying -> 프래그먼트 varying
* 버텍스 셰이더 입력 데이터 패킹
  + 각각 Array를 생성하는 경우
    - 이것도 괜찮은 방법
  + 합쳐서 Array생성하는 경우 – 번갈아가면서(x y z r g b a x y z…)
    - 가장 일반적임
  + 합쳐서 Array생성하는 경우 – 데이터별로 몰아서 (x y z x y z… r g b a r g b a)
  + 시험 출제 자주함 ~이러한 경우에는 어떤값을 넣어야 정상적으로나오는가?