2024-09-06-백민호

rex tutorials

viewer

- shader vsQuad(in vec3 position : 0, in vec3 normal : 1, in vec2 texcoord : 2, out PSIN vout)
  - vertex shader와 비슷한 개형을 가지고 있다.
  - gl_Position, vout.tex = texcoord으로 vsQuad의 out으로 넘겨준다
  - interface PSIN { vec2 tex; }
    - cg에서 배웠던 out vec2 tc 대신 interface 사용
  - GSLS
    - uniform : threads가 cpu에 받는 input, 일정(uniform)하고 read only, global하기 때문에 모든 shader에서 접근 가능
    - thread built-in output : gl*FragColor... input : gl_FragCoord... 주로 gl* 붙는다
      - built-in value는 varying type으로 thread마다 다르고 편집 가능하다
    - layout(location = ?) : CPU에 vertex data를 일정한 형태의 입력을 받아야 하는데, location meta data를 통해 vertex attribute를 구성할 수 있다. 우리가 VAO(또는 VBO) 만들 때 vertex의 구조와 순서를 정해준다. (cgut 참고)
    - static const size_t attrib_sizes[] = { sizeof(vertex::pos), sizeof(vertex::norm), sizeof(vertex::tex) };
    - interface block : gsls in, out, uniform 등을 그룹화 한것, struct의 gsls 버전
- shader psInvert(in PSIN pin, out vec4 pout)
  - 이전에 vs는 vertex shader의 줄임말이라 예상했는데 ps는? (fragment shader? fs?)
  - texel을 받아와서 pout으로 보낸다, pout.rgb를 통해 rgb값을 바꿀 수 있다
    - cg에서는 out vec4 fragColor(fragment shader에서)를 밖으로 보냈다
- program Invert { vs(440) = vsQuad(); fs(440) = psInvert(); };
  - Invert라는 program으로 vertex shader와 fragment shader에 위 두 함수를 붙여줌
  - 440?? : glsl 버전
- effect->bind("Invert"), effect->set_uniform("name", var_name)

    - effect : GL의 program 집합체, bind를 통해서 program을 붙여줄 수 있고, get&set_uniform을 통해서 uniform 변수를 다룰 수 있다. 추가적으로 image texture bind, vertex array를 통해서 primitive draw 연산을 수행한다.
- effect->draw_quads(); cg sample과 동일한 방식, color, depth buffer 비우고 vertex_array bind한 뒤 draw array 또는 draw element, GL_TRIANGLE_STRIP 사진은 네모

rgb2gray

- ntsc = vec3(0.299f, 0.587f, 0.114f); 사람이 인식하는 rgb 보정값

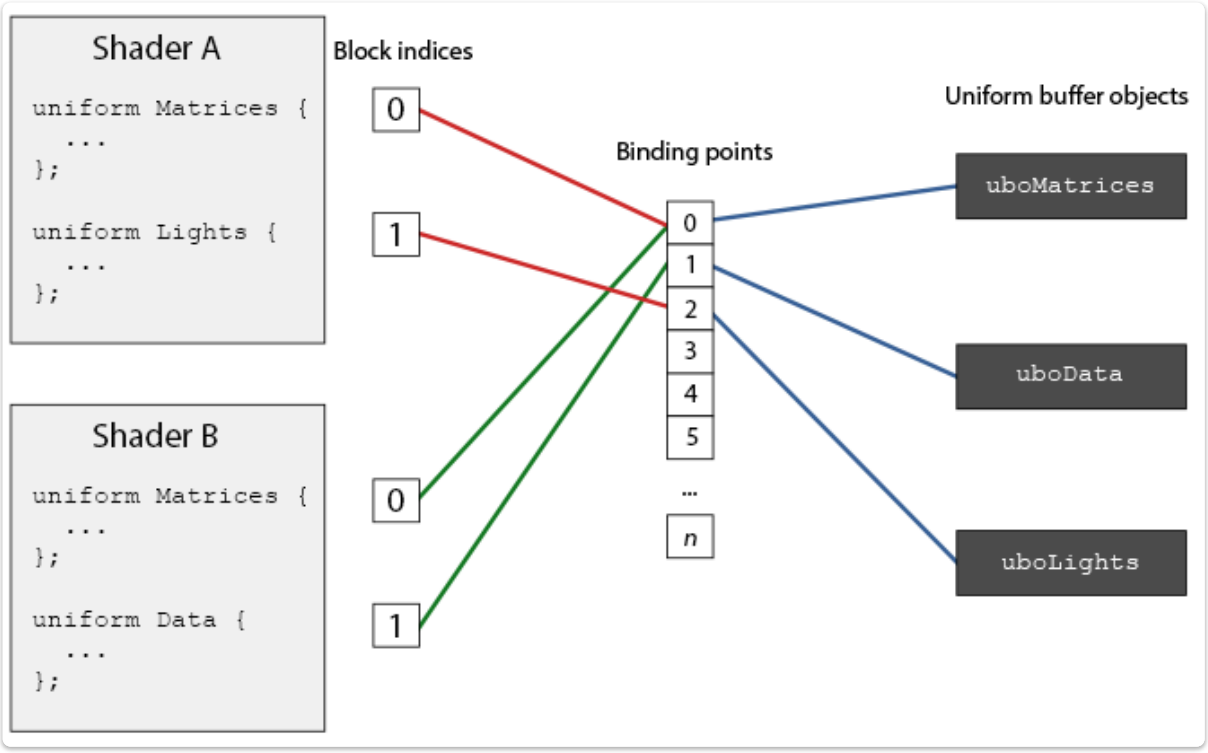
box-blur

- 2 pass를 활용한 blur, SRC -> TMP -> DST
- 커널rgb의 평균 값으로 픽셀 값 blur

moving-least-square

- 평면 이미지 tex 파일에서 추가적으로 원형 점 추가
- layout(std140, binding = 1), 기존에는 location을 이용하여 vertex attribute를 지정
    - interface처럼 uniform block, uniform buffer object를 만들어서 여러 shader program에 걸쳐 동일한 global uniform 변수 모음이다. 연관 uniform은 한번에 설정해야 함, glGenBuffers로 생성가능하고 GL_UNIFORM_BUFFER에 binding(userdata.cpp 참고)
    - std140 : 현재 정의된 uniform block은 지정된 memory layout을 사용하고 있다는 의미, 메모리에 적당한 offset이나 allign을 맞추기 위한 규칙들이 있는데 그 중에서 std140 layout을 쓰겠다는 것이다.
    - binding = ? : uniform block을 특정 binding point에 연결했다는 뜻

- 출처 : Learn OpenGL



  - 여러 uniform buffer를 여러 uniform point에 binding하여 여러 shader가 share

- cg course 학습하면서 두 오브젝트에 다른 효과를 주고싶었을 때 하나의 셰이더에 boolean을 이용하여 가지를 나눴는데, fx에서는 여러 shader program 분리
- on_dirty_image() : pp_image로 받은 이미지를 텍스쳐화? 한다. SRC에는 width와 height에 해당하는 img, DST는 빈 w, h 값을 가진 텍스쳐만 들어간다.
- fsDeform(in PSIN psin, out vec4 pout)
  : 1. 기준점의 개수(CONTROL_COUNT) 만큼 루프하면서 가중치 Wi를 기존의 Pi 위치와 계산하여 평균 Ps, Qs를 구함. pimv는 tex 좌표와 기준점 좌표의 차이 당연히 vertex와 가까운 기준점이 영향을 크게 미친다
  2.P0와 Q0 각각 P와 Q 위치에 P, Q 중앙점 차이 벡터, PWP와 PQ는 각각 PP, PQ 외적 *W*
  3.PQ 역행렬 PWP 곱하여 역방향 매핑을 위한 invM구하기, 현재 좌표 v를 변환한 새로운 좌표 f 계산 -> f = (v - Qs) * invM + Ps;

mipmap

- 2^n 사이즈의 n level로 이루어진 텍스쳐(eg. 256 x 256 -> 128 x 128, 64 x 64, 32 x 32 ...)들을 미리 만들어서 메모리에 올려 놓는 것으로 랜더링 속도가 향상된다.
- frame buffer object setting, depth나 cull 등등 T/F, bind(DST) : FBO에 텍스쳐 bind
- SRC texture 복사본으로 pout 생성(non-pot tex를 pot(2^n)로 바꿈, mipmap을 위해서!!)

- shader psBuildMipmap(in PSIN pin, out vec4 pout)
  - tc = ive2(gl_FragCoord.xy)<<1 : 픽셀 조정 2배씩 -> 해상도는 2배 감소
  - mipmap 생성 루프(render function)
    - FBO->bind(DST, 0, k) k-level rendering 대상으로 생성
    - DST->view(k-1, 1) 이전 레벨의 mipmap을 소스로 줄여나간다
- 이후 texelFetch(MIP, tc+offset, 0) 이전 레벨 텍스쳐 픽셀 값을 추출하고 유효 값들만 pout에 저장한다(offset은 현재 texel 주변 2 x 2(0,0), (1,0), (0,1), (1,1))
- create_control을 이용해서 키보드 입력 값에 대한 변화를 줄 수 있다.

minmax-mipmap

- 이전에는 주변 2x2 커널의 평균값을 이용 이번에는 minmax
- Framebuffer FBO, Effect effect, Texture DST(output) 추가로 Texture MMX(minmax mipmap 2 pass), Buffer GMX(global minmax uniform buffer)
- 1. 첫 init은 power of 2 없이 mipmap 생성
  - fbo에 mmx(texture), 0 level로 bind, init_mipmap program, sampler는 SRC
  - tc는 이전과 동일하게 픽셀 값 2배, ts는 텍스쳐 크기에 맞게
  - tc 주변 4개의 픽셀에 대하여 최솟값과 최댓값을 비교하여 pout에 저장
  - ivec2 t = tc+ivec2(0,0);
  - if(t.x<ts.x&&t.y<ts.y)
    { float z=texelFetch(SRC,t,0).x; pout=vec2(min(pout.x,z),max(pout.y,z)); }
- 2. level 값에 따른 mipmap 생성
  - for(k) loop를 통해 k-level의 mipmap texture를 fbo에 bind
  - 이전과 같은 방식에서 sampler를 MMX로 반복해서 mipmap으로 줄이기
  - effect uniform mmx는 k-1 level의 mipmap 이용
- 3. last level mipmap을 GMX로 copy
- 4. output DST로 랜더링

holegen

- on_mouse로 object interaction
- hole(x, y, size, valid?), b_selected(onClick method)
- render
  - tc = gl_FragCoord

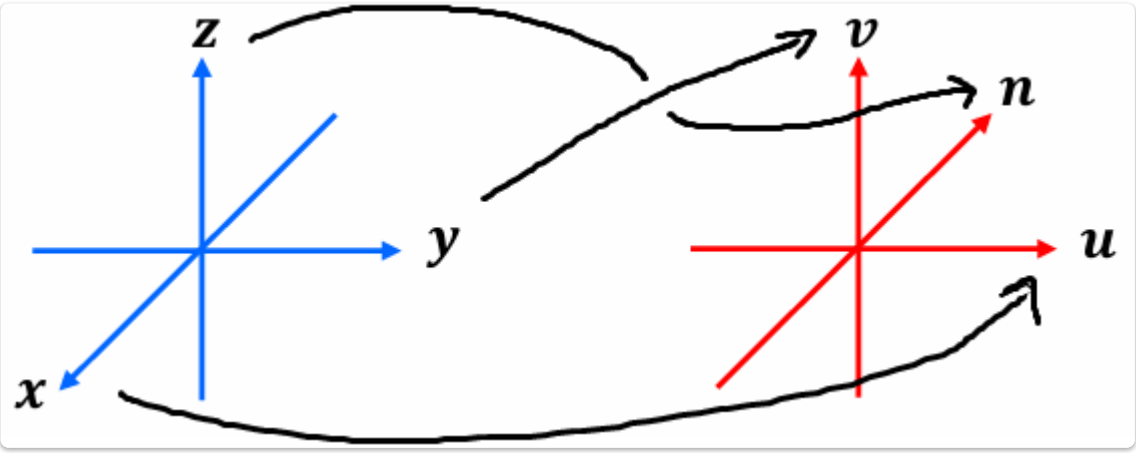
- uniform value : sampler2D SRC, vec3 hole(x, y, size)
- tex와 hole.xy 거리가 hole size보다 작으면 black or texelFetch(SRC, tc, 0)

pull-push-synth

- pull push synthesis??
  - mipmap처럼 다른 해상도를 이용해서 데이터를 보간하는 방법
- render
  - b_npot에 대해 npot texture -> pot texture 변환(padtopot)
    - ts : textureSize, 원본 texture의 w, h 저장
    - tc : 현재 픽셀 좌표(fragment 위치)
    - 현재 픽셀이 텍스쳐의 경계를 넘어가면 0,0,0,0으로 padding
  - pull phase
    - k level mipmap을 PULL texture에 저장(!b_npot 라면 k = 0 loop 1회 생략)
  - push phase
    - k = kn-2 부터 하는 이유는 kn-1은 가장 낮은 해상도라서 다룰 필요가 없기 때문
    - k-level의 PULL의 mipmap bind, PUSH k level에 넣어 줄 것(첫 loop 제외)
    - 추가적으로 k + 1 level의 PUSH texture bind(첫 loop에는 없으니 PULL texture)
    - tc 픽셀 값의 PULL texture를 가져와 pout에 저장, invalid pout에 대해 다음 처리
      - invalid한 것은 hole이고, 더 낮은 해상도를 가져와서 채우는 것
      - valid의 경우 PULL texture 그대로 pout 구멍에는 PUSH 채우기
      - PUSH(k+1) 주변 4개의 픽셀을 가져와서 pout에 더하기
      - 사각형 경계 검사, 정규화 등 진행
    - 마지막으로 PUSH 0 level DST에 draw_quads(!b_npot 경우 crop)

ubo

- uniform buffer object : uniform data를 저장하기 위한 buffer object, 다른 program 사이에서도 share 가능
  - struct 형태로 구현된 uniform data를 일괄적으로 set 가능
  - layout(std140, binding=0) uniform SPHERE{ sphere_t sphere; }; (GLSL)

    - gl::Buffer* ub_sphere = effect->bind_uniform_buffer("SPHERE");
      ub_sphere->set_data(sph_data);
    - layout(std140, binding=1, row_major) uniform MODEL{ model_t model; };
        - binding : GPU의 메모리에 저장될 위치, 각 uniform block 충돌 x
        - row_major :저장 방식에 대해 default column major, 행렬 쓸 때 cpu와 다른 방식
- effect->draw_quads() 가 아니라 VAO->draw_elements(0) + vertex shader 차이점
    - vertex에 대해 wpos, view_projection_matrix 추가 처리 필요
- view-projection matrix
    - model -> world -> view(camera가 원점) -> clip
    - {0,1,0,0, 0,0,1,0, -1,0,0,1, 0,0,0,1}, x -> y, y -> z, z -> -x에 축 맞추기
    - 

ssbo

- shader storage buffer object (openGL wiki)
    - buffer를 이용한 interface block의 일종
    - GLuint ssbo;
      glGenBuffers(1, &ssbo);
      glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo);
    - ubo와 차이점
        - ubo는 16KB, ssbo는 128KB까지 gpu memory를 할당할 수 있다
        - writable(even atomically), ubo는 읽기 전용 변수들을 저장, 읽고 쓰는데 incoherent memory accesses를 사용하기 때문에 memory barriers 필요
        - memory layout이나 binding qualities에 대한 special qualifier가 있음(eg. std430)
    - shader
        - struct image_data_t { ivec3 count; int pad; };

        - layout(std430, binding=1) buffer HIST { image_data_t data[]; };
        - storage block var를 사용하기 위해 atomic function 필요
            - atomicAdd, Min, And etc...(int, uint type에서만 작동한다)
- render
    - 1. pp_SRC -> SRC -> DST에 image copy
    - 2. SRC rgb 값을 HIST에 누적 연산
        - b_atomic_sync -> atomicAdd : 항상 일정한 값 출력
        - !b_atomic_sync -> 단순 += 연산 -> 값이 일정하지 않고 작음(덮어 써짐)
        - compute shader와 fragment shader(openGL wiki)
            - compute shader : GPUI 범용 계산을 수행하는데 최적화, 무언가 그리기 보다는 히스토그램 계산 같은 작업을 수행하고 랜더링 과정이 아니기 때문에 frament shader가 필요 없게 된다. (FPS나 시간 변화 차이 확인하기)
            - ComputeShader는 work group(x, y, z)이 존재하는데 work group에 여러 threads가 존재할 수 있다.
            - 실행 할 때, 몇 개의 work groups, 몇 개의 threads를 실행할 지 결정해야 한다. 이렇게 구분하는 이유는 thread를 가변적으로 설정할 때가 있고, 동일 work group의 thread는 동시 실행, 언제 실행하든 같은 output을 내야하기 때문이다.(work group 내의 shared memory와 synchronizing)
            - tc = ivec2(gl_GlobalInvocationID.xy);
                - gl_WorkGroupID : 현재 work group id
                - gl_WorkGroupSize : work group 내 thread 수
                - gl_LocalInvocationID : work group 내 현재 thread id
                - gl_GlobalInvocationID = gl_WorkGroupID * gl_WorkGroupSize + gl_LocalInvocationID

minmax-compute

- minmax mipmap 연산을 compute shader 통해 parallel하게 계산
- render
    -