* 텍스처 매핑
  + 필요한 데이터 : 이미지(jpg,bmp 압축 포맷 등), 매핑 위치
  + OpenGL은 uv 축이 왼쪽아래가 0,0 이다
  + u, 1-v 로 쓰면 왼쪽위가 0,0이 됨
  + uv 의 유효 범위는 0~1이다
  + 텍스처 샘플링
    - 컬러를 결정할 때 필요한 Data를 가져오는 것, 컬러만 결정하는 건 아님
    - 일반적으로 프래그먼트 셰이더에서 이루어짐
  + 텍스처 생성
    - 전체적인 순서는 VBO와 비슷하다
    - glGenTextures(1,&gTextureID); 생성후 안쓰면 날려야함
    - glBindTexture(GL\_TEXTURE\_2D, gTextureID);
    - glTexImage2D(GL\_TEXTURE\_2D,0,GL\_RGBA,8,8,0,GL\_RGBA,GL\_UNSIGNED\_BYTE,checkerboard); 데이터가 이동되는 부분
    - glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_NEAREST);  
      GL\_MAG\_FILTER,GL\_LINEAR(보간)
  + 셰이더에 if문이 존재하면 2배 느려짐
  + 다중 텍스처 사용
    - GL\_TEXTURE\_2D는 설정 가능한 텍스처가 최소 80개임
    - 텍스처를 설정하기 위해 bind전에 active가 필요
  + 단일 텍스처 애니메이션
    - 다중 텍스처를 이용하면 텍스처 switch 발생으로 효율성이 떨어진다,캐시 효율성도 떨어짐
    - 하나의 텍스처에 여러장의 텍스처를 합쳐서 그리는 방식이 switch 발생이 없고, 캐시 효율성이 높다
* 버텍스 셰이더 응용
  + 프레그먼트 셰이더 기반의 한계
    - 특정부분은 그리고 안그리는 기법이 핵심인데 안그리는 부분 뒤쪽에 뭔가가 있어서 그려야 할 때 취소 작업이 필요하다.  
      (Discard : pixel 출력을 안함, 분기문(if) -> 캐시 효용성 망가뜨림, 가급적 쓰지 말 것)
    - 엄청난 계산량
    - 쉐이딩이 어려움
* 프레임버퍼
  + 화면에나타낼 그림의 정보가 저장된 메모리 영역
  + 보통 GPU 메모리에 할당됨
  + 윈도우 os는 각 윈도우마다 프레임 버퍼가 할당되며 다른 OS의 경우 대동소이함
  + 메인 프레임버퍼가 최종적으로 모니터와 연결됨
  + 화면 모드
    - 윈도우 모드: 윈도우 모드 경우 다른 윈도우 및 바탕화면 등의 프레임버퍼와 합성이 된 후 최종 결과 출력, 속도가 느리다
    - 전체 모드: 전체모드이면 해당 어플만 프레임버퍼 전체화면에 표시, 하나만 그리면 되니까 속도가 더 빠름
  + 윈도우 프레임 버퍼에 Open GL이 그림 그리도록 하는 것은 복잡함
  + glut에서 간략하게 사용할 수 있게 해줌
    - glutInitDislayMode(GLUT\_DOUBLE|GLUT\_RGBA) //더블 버퍼링, 픽셀당 RGBA
    - glutInitWindowSize(1024,1024) // 창 사이즈
    - glutCreateWindow(“Tutorial”) // 윈도우 제목
  + 더블 버퍼링
    - glutSwapBuffers(); //Open GL이 back 버퍼에 그린 것을 frons 버퍼로 swap함
  + 메인 프레임버퍼는 오픈지엘 윈도우 생성시 자동으로 0번 location에 바인드됨
  + 지금까지 디폴트인 0번으로 출력했음
  + 화면에 출력이 안되는 버퍼를 위한 기능: Framebuffer Object: FBO
* 프레임 버퍼 오브젝트
  + FBO 자체는 껍데이이고, 내용을 따로만들어서 attach 해야 함
  + Rgba, depth 따로 생성해서 attach해야함
  + Color buffe에 Texture attach 할 수 있음
    - GLuint textureId; glGenTextures(1,&gFBOTexture);
    - glBindTexture(GL\_TEXTURE\_2D,gFBOTexture);
    - glTexParameterf();
    - glTexImage2D(GL\_TEXTURE\_2D,0,GL\_RGBA8,512,512,0,GL\_RGBA8,GL\_UNSIGNED\_BYTE,0);
  + depth buffer
    - depth buffer의 경우 따로 attach 하지 않아도 기본 depth buffer를 기반으로 시작
    - 하지만 depth 정보를 덮어버리기 때문에 따로 사용하는 것이 안전
  + Render buffer 생성
    - GLuint depthBuffer;
    - glGenRenderbuffers(1,&depthBuffer);
    - glBindRenderbuffer(GL\_RENDERBUFFER,depthBuffer);
    - glRenderbufferStorage(GL\_RENDERBUFFER,GL\_DEPTH\_COMPONENT,512,512);
    - glBindRenderbuffer(GL\_RENDERBUFFER,0);
  + FBO 생성
    - glGenFramebuffers(Glsizei n, GLuint\* ids)
    - glGenFramebuffers(1,&FBO0);
  + FBO 내용 채우기
    - glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, m\_FBOTexture0,0);
    - glFramebufferRenderbuffer(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_RENDERBUFFER, m\_RBDepth0);
  + 에러 체크
    - GLenum status = glCheckFramebufferStatus(GL\_FRAMEBUFFER);
    - If(status != GL\_FRAMEBUFFER\_COMPLETE)
  + glBindFramebuffer(GL\_FRAMEBUFFER, 0); 반드시 이걸 해주어야 한다. 다시 메인 프레임 버퍼를 그린다
  + glViewport(Glint x, Glint y, Glsizei width, Glsizei height)
    - 프레임버퍼 안의 그림을 그릴 영역을 설정하는 함수
    - 지금까지 default로 메인프레임 버퍼 크기만큼으로 설정됨