* 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