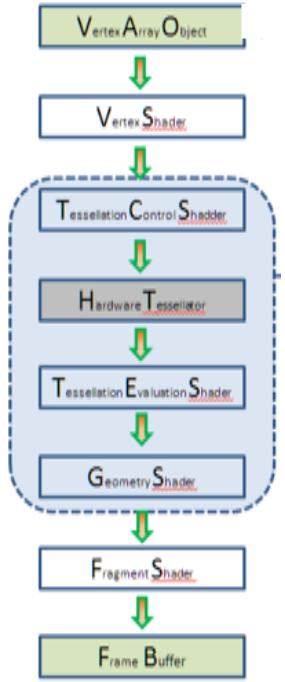
Vertex Shader-based Effects

- Topics
  - Geometric transformations
  - Surface Deformation
  - Displacement mapping
  - Wave
  - Per-vertex illumination

Vertex Processing: Ideas

- A geometric mesh can be transformed by performing vertex-by-vertex processing using a vertex shader
  - either **globally** (uniform transformation)
  - or **locally** (non-uniform transformation)



Vertex Based Graphics Effects: Ideas

- Capture the incoming position:

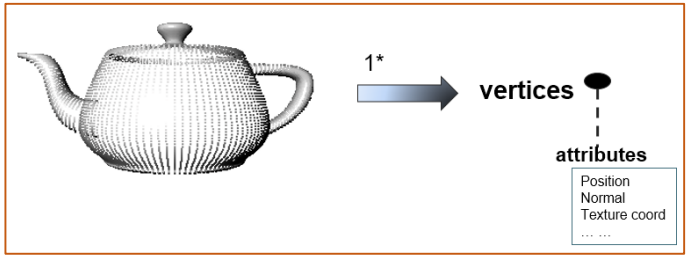
  vec4 Pos=gl_Vertex;

- Transform the captured position:

  Pos.xyz = … … ;

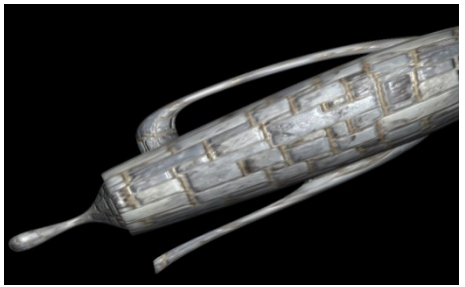
- Send the transformed position to clipping space:

  gl_Position =gl_ModelViewProjectionMatrix*Pos;



Per-Vertex Geometric Transformation

- Types of geometric transformations
  - Affine:
    - Flat ➔ flat
      - Translation, scaling, rotation, shearing, …
  - Non-affine:
    - Flat ➔ curved



Localized Transformations

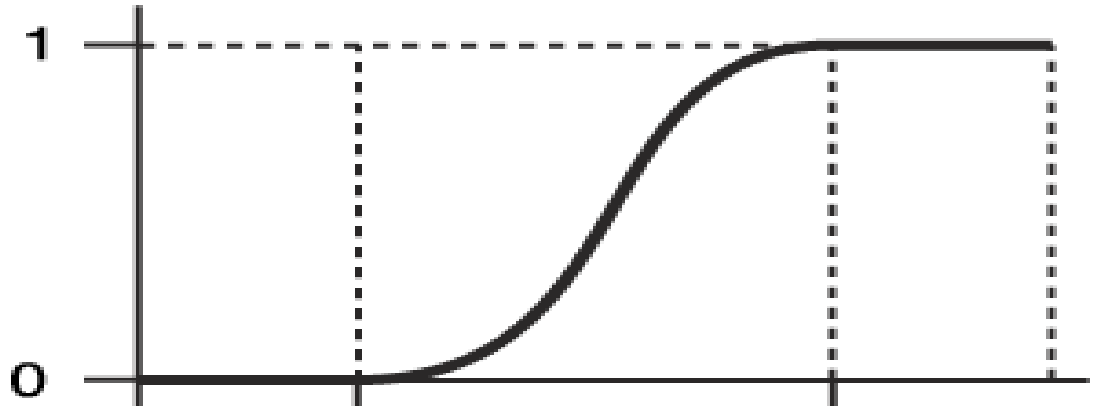
- Apply different transformations differently for different parts of a geometric objects

```
if(Pos.y > 21.0 && ...)
{
    //Apply transformations
    ... ...
}
```

Smoothed Local Transformations

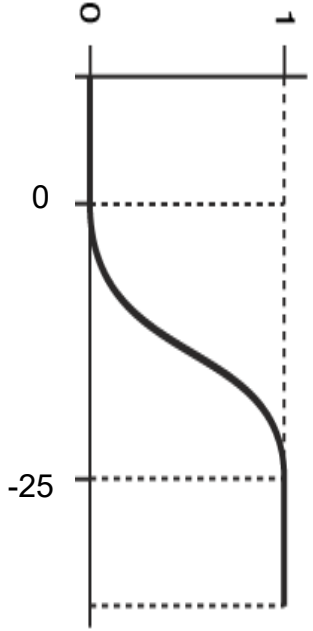
- Control transformation scope using GLSL smooth unit step function

smoothstep(a, b, x);



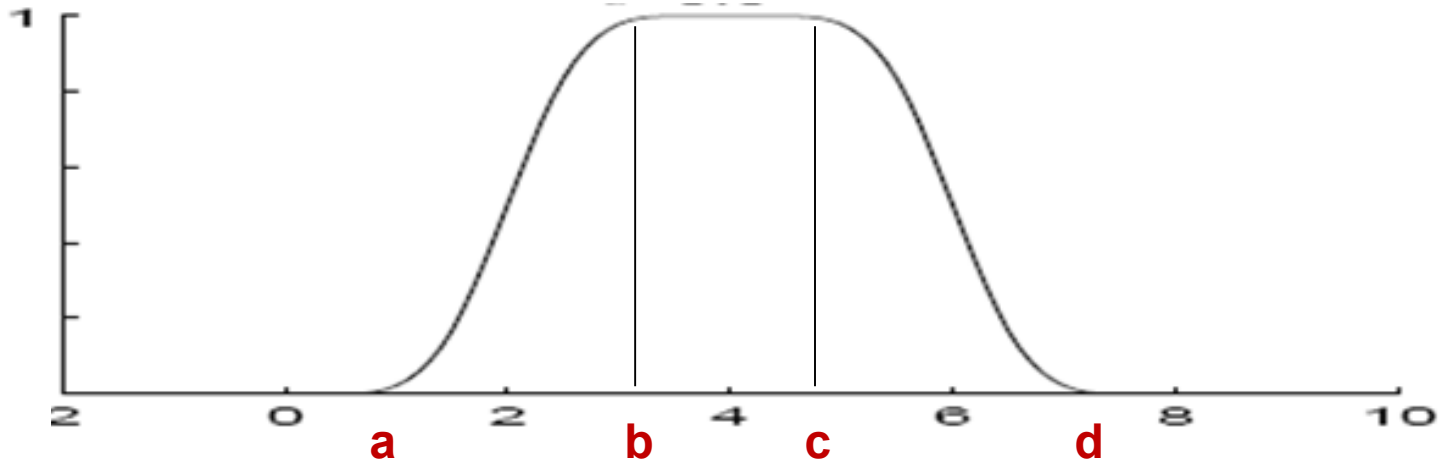
Smoothed Local Transformations

Pos.xyz += scale*gl_Normal*(1.0-smoothstep(-25.0, 0.0, Pos.y));



Smoothed Local Transformations

```
float softIntveral(float a, float b, float c, float d, float x){
    return smoothstep(a, b,x)-smoothstep(c, d,x);
}
```

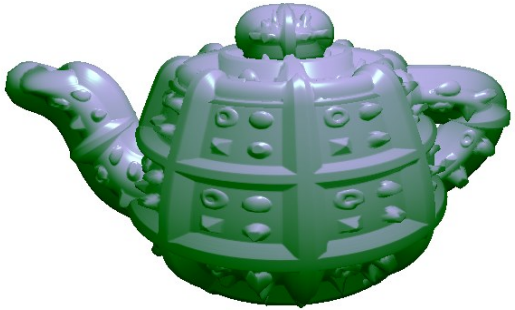


Smoothed Local Transformations

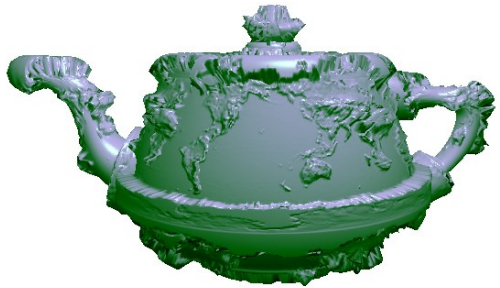
```
float s=scale*softIntveral(-10.0, -5.0, 5.0, 10.0, Pos.x);
Pos.xyz  += s* normalize(gl_Normal);
```

Displacement mapping: Texture-Based Deformation

- Often referred to as displacement mapping by interpreting a texture as a height map.



```
… …
vec4 inPos = gl_Vertex;

float H=texture2D(HeightMap, gl_MultiTexCoord0.xy).x;

inPos.xyz +=S*H*normalize(gl_Normal.xyz);

gl_Position = gl_ModelViewProjectionMatrix * inPos;
… …
```



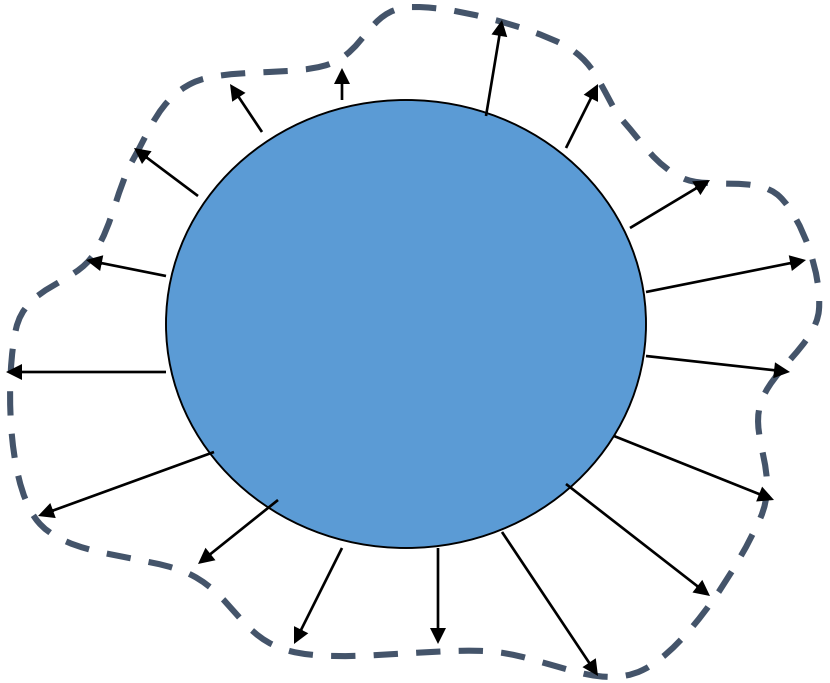
VS-based animation

- A vertex-based animation can be implemented in the following way
  - Input a timer to vertex shader
  - Specify per-vertex transformation as a function of time
    - To change input vertex position, orientation and appearance

Example: Pulsating animation

- Change the position of each vertex by scaling it outward or inward along the normal direction

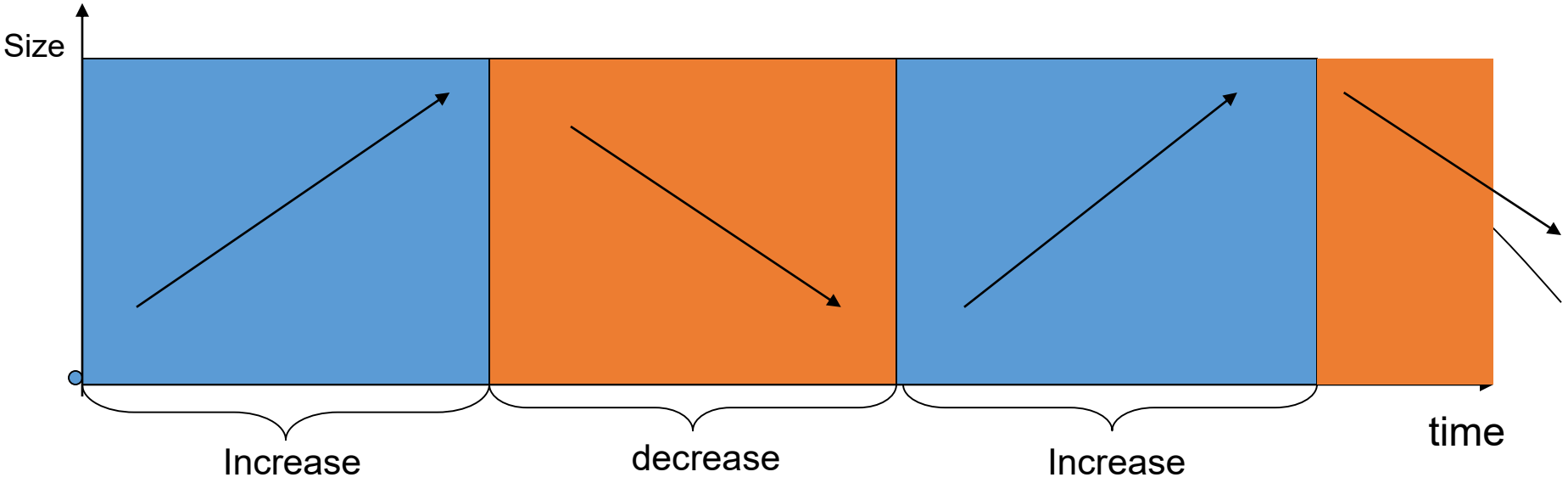
- Update the displacement amount over time
  - Uniformly
  - Or non-uniformly



Define displacement amount

- Specify the displacement amount as a sine function of time:

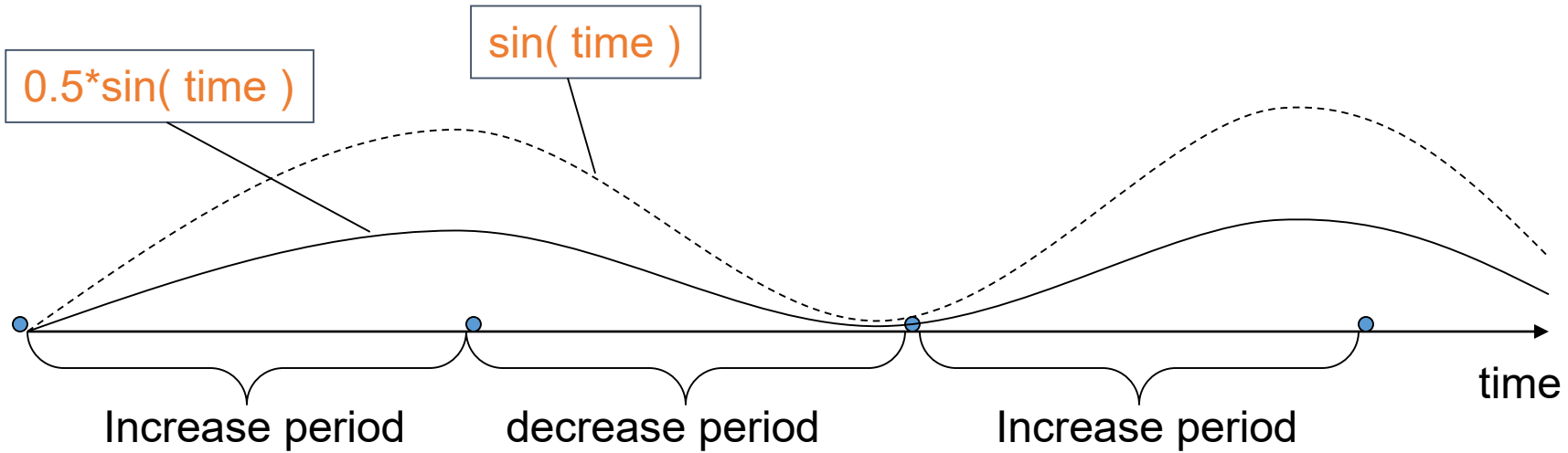
displacement_size= sin(time)



Control the Magnitude

- Can be easily achieved by introducing a parameter **s**: to control the magnitude of growth:

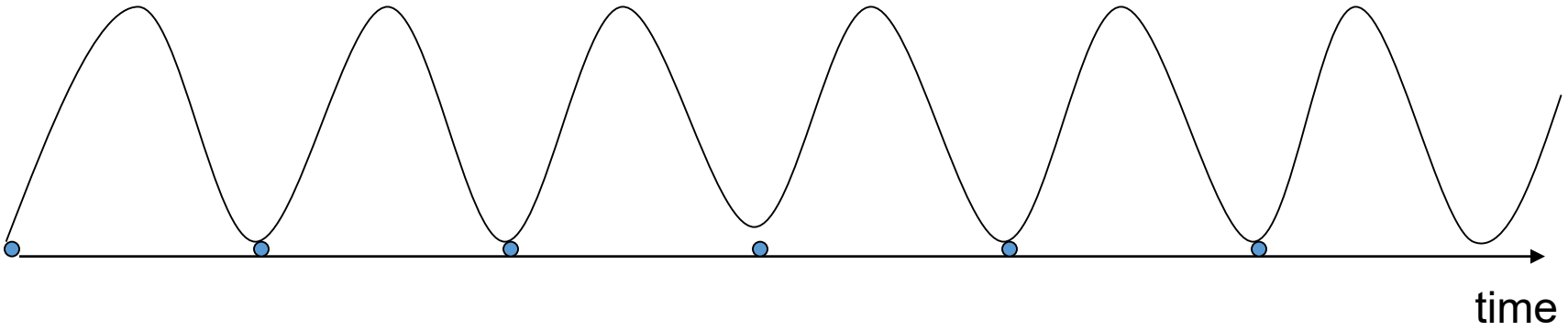
  displacement_size= **s***sin(time);



Control the Frequency

- Can be achieved using a a paramter **freq**:

  displacement_size= **s*** sin (**freq** * time);



Vertex Shader

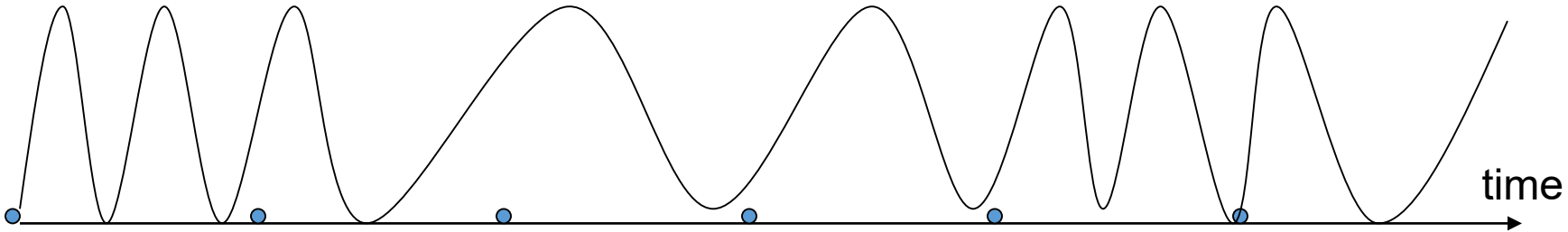
```
... ...
uniform float time;
uniform float mag;
uniform float freq;
void main( void )
{
    //get the input vertex position:
    vec4 inPos = gl_Vertex;
    //define the pulsating scope:
    float disp= mag * sin ( freq * time);
    //get the normal vector:
    vec3 N=normalize(gl_Normal);
    vec3 N=normalize(gl_Normal);

    inPos.xyz += disp * N;

    gl_Position =
        gl_ModelViewProjectionMatrix*inPos

}
```

Non-uniform Vertex Transformation

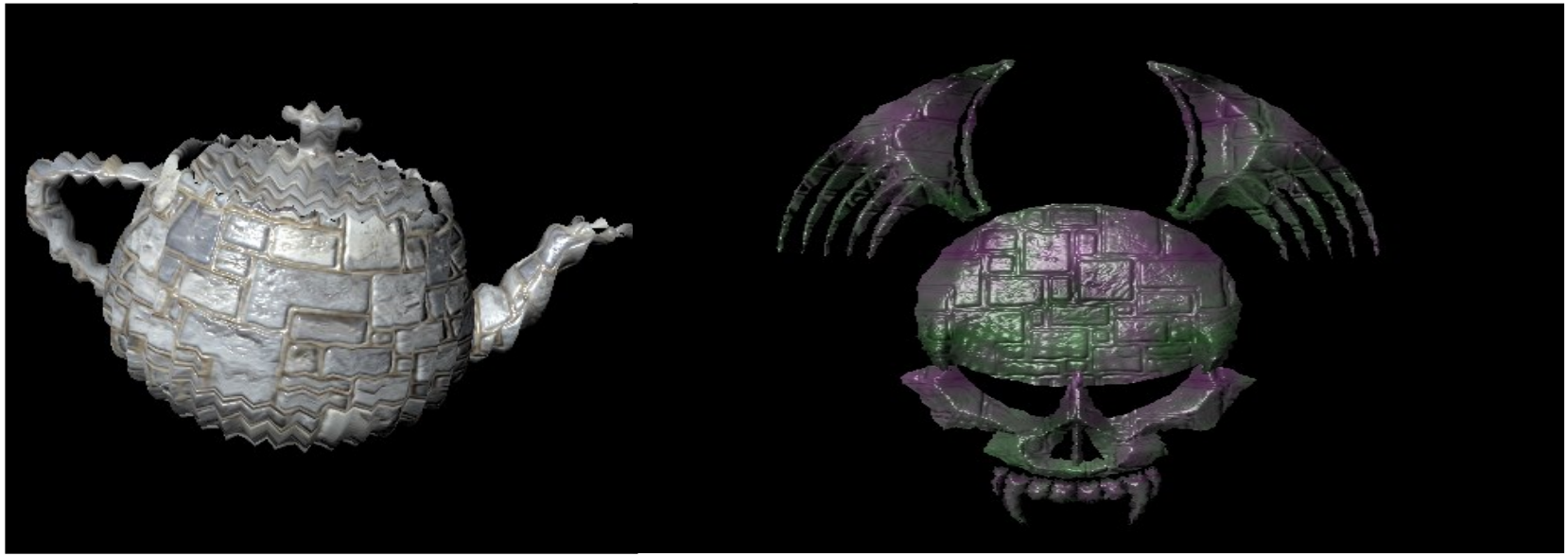
- Can perturb different vertex differently by introducing parameters relating to the vertex position:

disp= mag*sin(freq(x,y,z)*time)+g(x,y,z));

where x,y,z are coordinates of the input vertex position

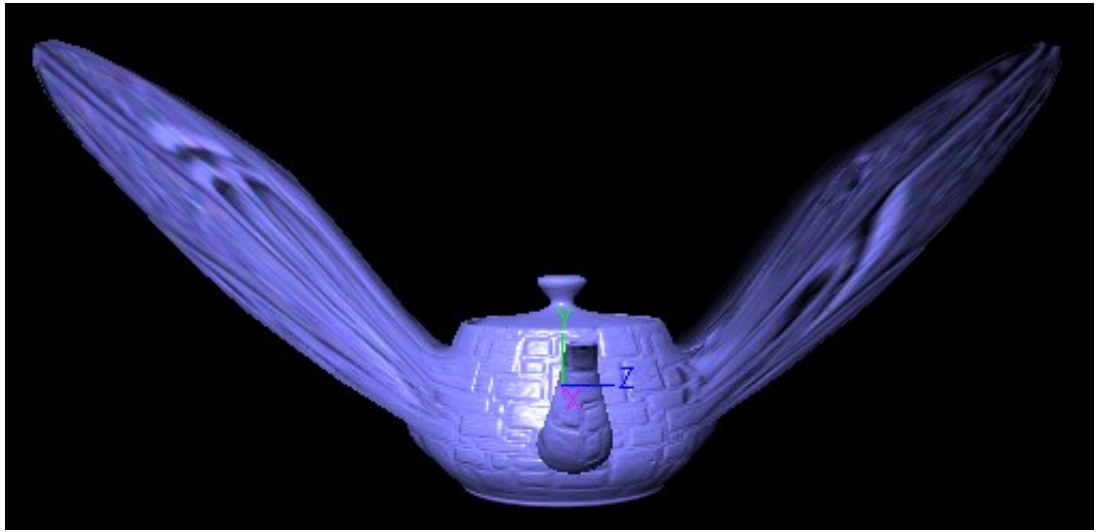


Demos



Animation: A Flying Teapot

- Skinning effect can be achieved with a collection of localized transformations



Animation: A Flying Teapot

- Creating the wings using smoothly controlled local transformations
- Rotate the wings using locally controlled rotations

```
float Angle =Mag*sign(Pos.z)* sin(freq*time);
```

Questions?