Vertex Based Animation

Vertex Based Animation: Idea

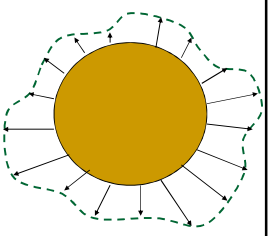
- Specify the change of an object in position, orientation and shape as a function of time
- Implemented in per-vertex manner

Vertex Shader Based Geometric Transformation

- A geometric transformation can be easily implemented in vertex shader
  - Affine:
    - Translation, scaling, rotation, shearing, …
  - Non-affine:
    - Such as non-linear transformations

Example: Pulsating objects

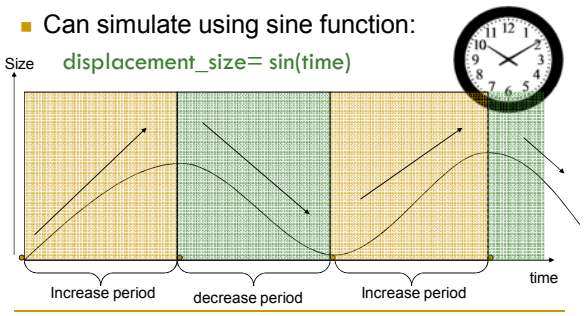
- Idea
  - Change the position of each vertex by pushing or pressing it along the normal direction
  - The displacement must be updated over time



Create an animation as a function of time

- Can simulate using sine function:

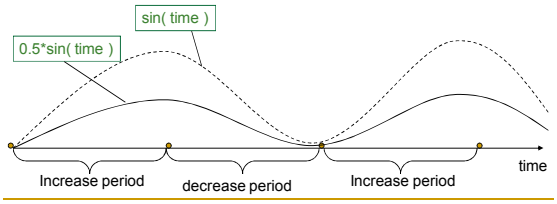
displacement_size= sin(time)



Control the amplitude

- Introduce a paramter **s**:

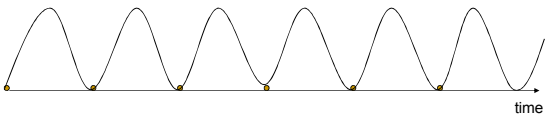
displacement_size= **s***sin(time);



Control the frequency

- Introduce a paramter **freq**:

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



Vertex shader

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

Vertex shader

```
//change vertex position in the world space:
trPos = trPos + vec4(disp*N, 0);

//transform the altered vertex position into clipping space:
 gl_Position = glModelViewProjectionMatrix *trPos;
//ftransform() cannot be used, which only applies to unaltered
//vertex position
… …
… …
```

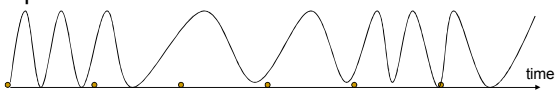
Demo

Perturb different vertices differently using their position

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

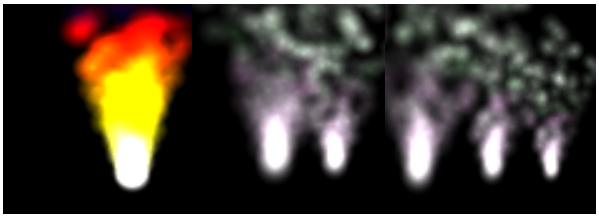
  disp= ampl*sin(f(x,y,z)*freq*time)+g(x,y,z));

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



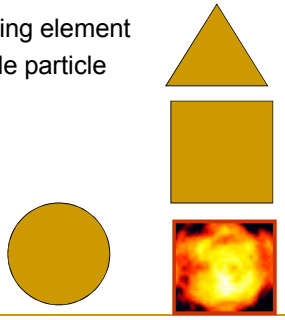
Demo

Example: Particle systems



What is a particle

- A dynamically moving element
- Attributes of a single particle
  - Shape
  - Texture
  - Position
  - Velocity
  - Mass
  - Life span
  - Path



Modeling the behavior of a single particle

- All attributes of the particle are functions of time
  - **Position**
    - move along certain path
    - Determined by all the forces acting on the particle
      - Force →Acceleration → Velocity → Position
  - **Colour**
    - Change along time
    - Depending the effects to be created
  - … …

Create particles

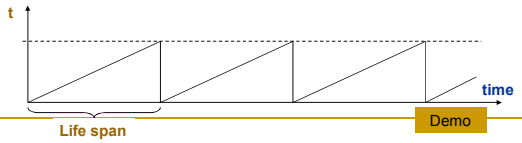
- Load an array of simple geometric objects, such as an array of quadrilaterals
  - Such as QuadArray.3ds used in RenderMonkey samplers
    - QuadArray.3ds consists of a hundred quads, each of which is a simple (-1,-1) to (1,1) quad. The quads are differentiated by their z-value, which has a ranging from 0 to 1.
  - Use the positions of particles as parameters to specify how to distribute the particles in the space

Modeling the behavior of a single particle

- Life span

```
uniform float lifeSpan;
float t = mod (time, lifespan );
```

- The life span of a particle can also be modelled using function fract( )

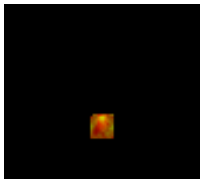


Demo

Modeling the behavior of a single particle

- Particle with constant velocity
  - Move along a straight line
  - Position of a particle can be calculated in the following way

```
vec4 PPos = gl_Vertex;
float t = mod (time, lifespan );
PPos.xyz  +=  velocity * t;
```

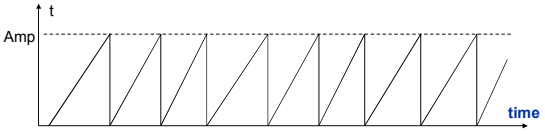


Demo

Modeling the behavior of a single particle

- Control the lifetime and birth-death frequency of a particle

```
uniform float Amp;
uniform float freq;
float t  t = Amp*mod(freq*time, lifeSpan);
```



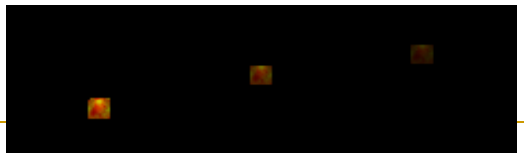
Modeling the behavior of a **single particle**

- Update particle colour in Vertex shader
  - Vertex shader:

```
float dist = length(displacement);
float maxDist =  length(lifespan*velocity);
float fadingRate = (1 -  dist/maxDist);
```

  - Pixel shader:

```
return texture2D( baseMap, Texcoord )*fadingRate;
```



Modeling the behavior of a **Particle system**

- Spread particles.
  - For example

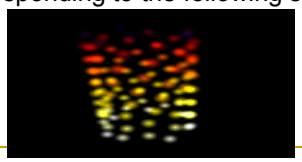
```
PPos.x = particleSpread * cos(300 * Pos.z);
PPos.z = particleSpread * sin (300 * Pos.z);
PPos.y = particleSystemHeight;
```

  will corresponding to the following shapes:

Particle systems: Spread particles using **t**

- With
  - PPos.x = particleSpread *cos(300 * Pos.z);
  - PPos.z = particleSpread *sin (300 * Pos.z);
  - PPos.y = particleSystemHeight * **t**;

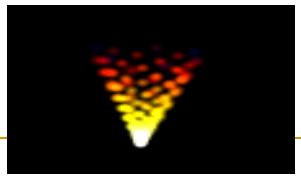
  will corresponding to the following shapes:



Particle systems: Spread particles using **t**

```
PPos.x = particleSpread * t * cos(300 * Pos.z);
PPos.z = particleSpread * t * sin (300 * Pos.z);
PPos.y = particleSystemHeight*t;
```

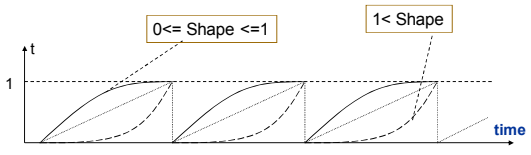
will produce the following particle shape:



Particle systems: Control the particles' velocity

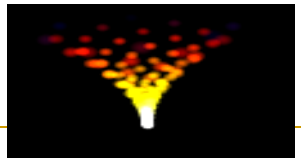
- float s = pow(t, particleSystemShape);

$$s = t^{shape}$$



Particle systems: Spread particles

- More shape can be generated when replacing t with s in the above equations
  - PPos.x = particleSpread ***s*** cos(300 * Pos.z);
  - PPos.z = particleSpread ***s*** sin (300 * Pos.z);
  - PPos.y = particleSystemHeight***t***;



Particle systems: Particle dressing

- Billboard the quads.
  - Set xy-plane of each quad such that it always faces the viewer
- How to:
  - Find the inverse of ModelView Matrix

```
uniform mat4  view_inverse_matrix;
```

  - Find direction of x-axis and y-axis

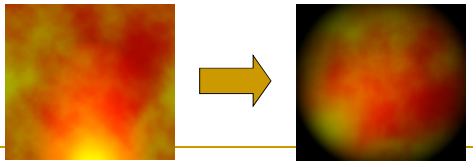
```
vec3 ViewSpX = view_inverse_matrix[0].xyz;
vec3 ViewSpY = view_inverse_matrix[1].xyz;
```

  - Reset particle shape orientation:

```
PPos += particleSize * (    gl_Vertex.x * ViewSpX
                          + gl_Vertex.y * ViewSpY );
```

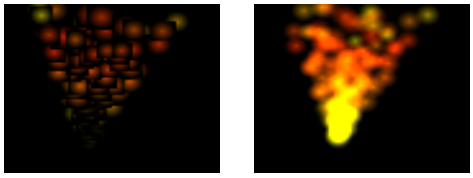
Fade the texture colour further

```
float x=TexCoord.x; float y=TexCoord.y;
float circle = radius*radius - ( (x-0.5)*(x-0.5)
                                 + (y-0.5)*(y-0.5) );
float shade = 2/(1+exp(12*circle));
return (1- shade)*tex2D( baseMap, Texcoord );
```



Alpha transparency blending

- Set GL_BlendEnable to be true to enable alpha transparency blending



Without alpha-blending

With alpha-blending