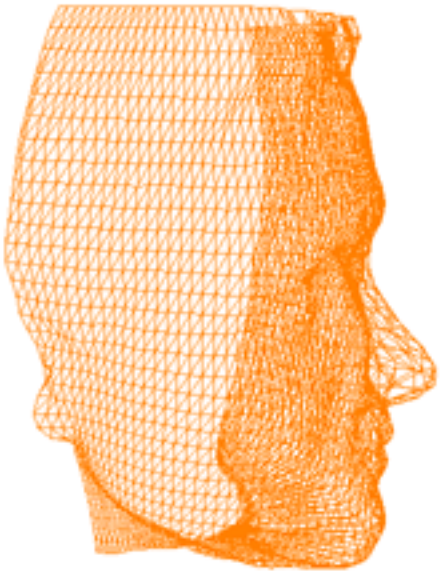
COSC422 Advanced Computer Graphics

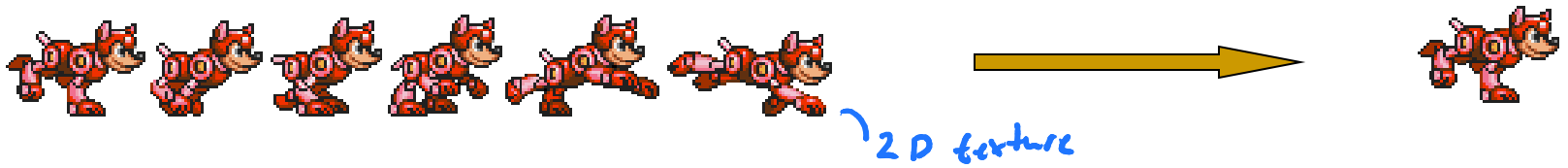


3 Sprites and Particle Systems

Semester 2
2021

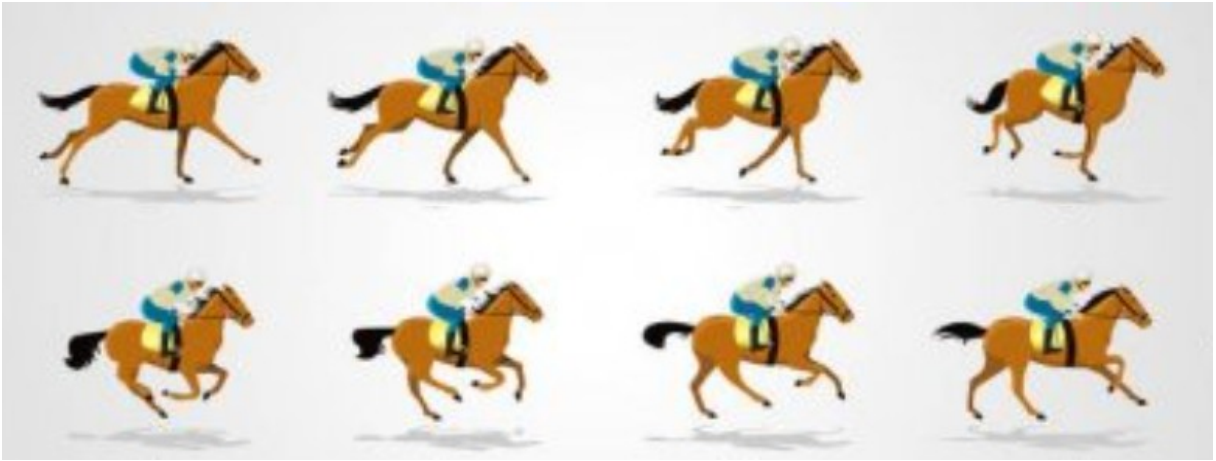
R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Sprites



- Sprites are **screen-aligned two-dimensional textures** that are added to a rendered scene.
- Rotational and shear transformations are not generally applied to sprites.
- An animation is generated by displaying several sprites in quick succession.
- Sprites are extensively used in arcade games.

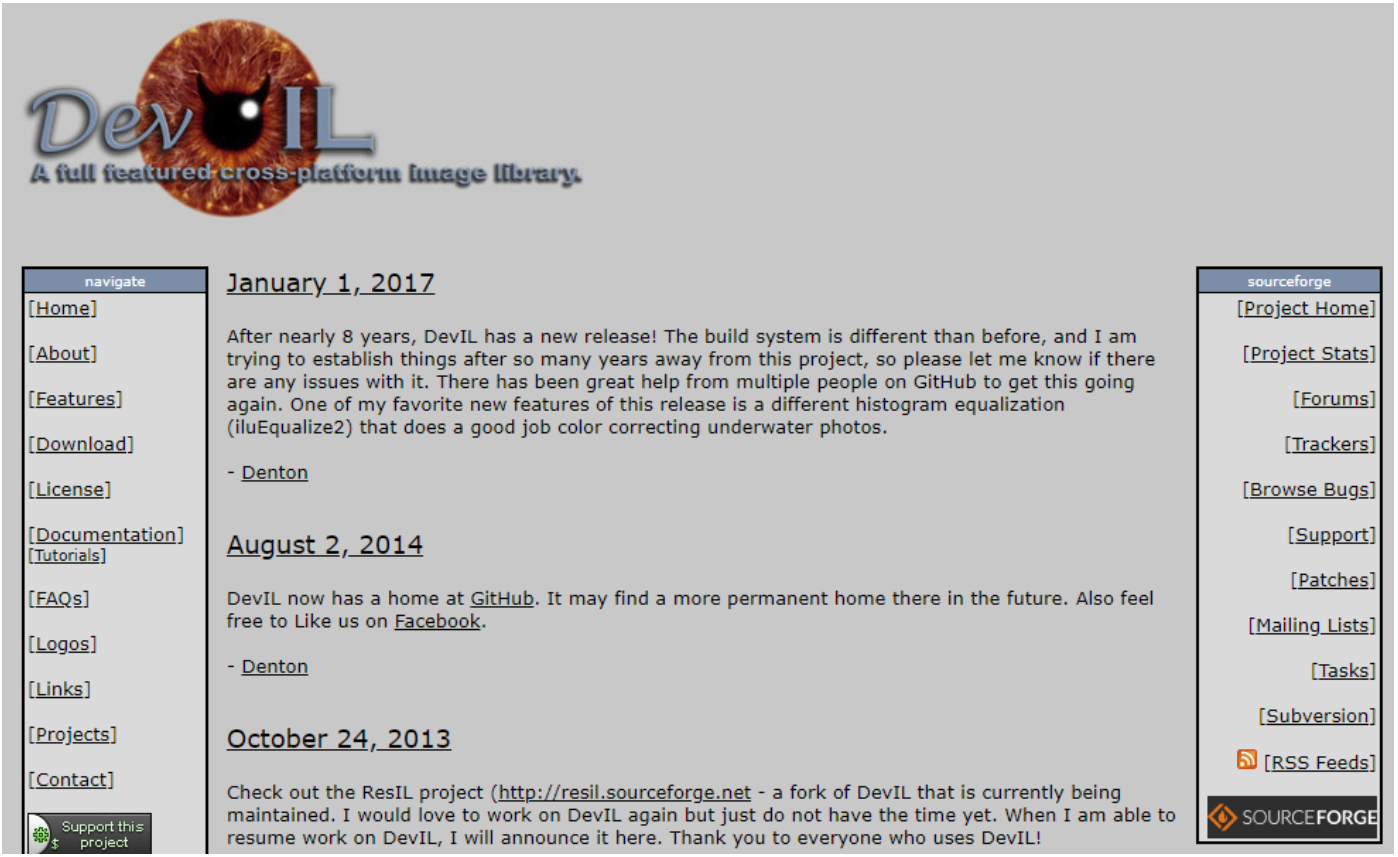
Sprite Sheets



DevIL Library

We require an image library that can load images in different formats (bmp, jpg, png etc.) as textures.

http://openil.sourceforge.net/



Texure Loader: Code

DevILTest.cpp

```
#include <IL/il.h>
using namespace std;

GLuint texId;

void loadGLTextures()
{
    ilInit();
    ILuint imageId;
    ilGenImages(1, &imageId);               // Create an image id similar to glGenTextures()
    ilBindImage(imageId);                   // Binding of DevIL image name
    ilEnable(IL_ORIGIN_SET);                // Enable change of origin of reference
    ilOriginFunc(IL_ORIGIN_LOWER_LEFT);     //This is necessay since the default location is upper-left

    if (ilLoadImage("Colors.png"))          //if success
    {
        int imgWidth = ilGetInteger(IL_IMAGE_WIDTH);
        int imgHeight = ilGetInteger(IL_IMAGE_HEIGHT);
        /* Convert image to RGBA */
        ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
        glPixelStorei(GL_UNPACK_ROW_LENGTH, 0);
        glPixelStorei(GL_UNPACK_SKIP_PIXELS, 0);
        glPixelStorei(GL_UNPACK_SKIP_ROWS, 0);

        //OpenGL Texturing
        glGenTextures(1, &texId);
        glBindTexture(GL_TEXTURE_2D, texId);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, imgWidth, imgHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, ilGetData());
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        cout << "Texture successfully loaded. Width = " << imgWidth << " Height = " << imgHeight <<endl;
    }
```

Sprite Frames

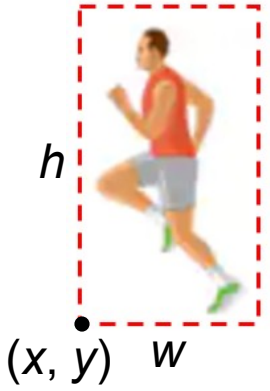
→ 14 Frames

Frame	x	y	w	h
0	24	200	122	180
1	215	200	135	180
2	395	200	155	180
3	589	200	151	180
4	782	200	141	180
5	993	200	115	180
6	1183	200	115	180
7	1372	200	131	180
8	30	8	134	180
9	213	8	146	180
10	398	8	155	180
11	605	8	127	180
12	808	8	100	180
13	995	8	103	180

- each bounding box (from 14 frames)

- require 14 frames - bounding box

- The program that runs this only runs one point.



Extracting Sprite Frames

```
if (ilLoadImage("Runner.png"))
{
    imgWidth = ilGetInteger(IL_IMAGE_WIDTH);
    imgHeight = ilGetInteger(IL_IMAGE_HEIGHT);
    ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
}

glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glPixelStorei(GL_UNPACK_ROW_LENGTH, 0);
glPixelStorei(GL_UNPACK_SKIP_PIXELS, 0);
glPixelStorei(GL_UNPACK_SKIP_ROWS, 0);

glGenTextures(1, &texID);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texID);
imgData = new BYTE[frameWid * frameHgt * 3];
ilCopyPixels(frameX, frameY, 0, frameWid, frameHgt, 1, IL_RGB, IL_UNSIGNED_BYTE, imgData);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, frameWid, frameHgt, 0, GL_RGB, GL_UNSIGNED_BYTE, imgData);

glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
delete imgData;
```

Point Sprites (OpenGL 4)

- Even though a point is represented by a single vertex (*x, y, z,* 1), it can be displayed as a **square** region on the screen with a user specified size (in pixels)
- The only transformation that will have any effect on a point is translation.
- A point's size (in pixels) can be specified by
  - Calling **glPointSize(size)** in the application, or,
  - Assigning a value to **gl_PointSize** in the vertex shader. For this, the state GL_PROGRAM_POINT_SIZE must be enabled.
  - This is not a scale transformation of a point

Point Sprites (OpenGL 4)

- Application enables the following states:

```
glEnable(GL_POINT_SPRITE);
glEnable(GL_PROGRAM_POINT_SIZE);
...
glDrawArrays(GL_POINTS, 0, 1);
```

enables it eg a quad fragment

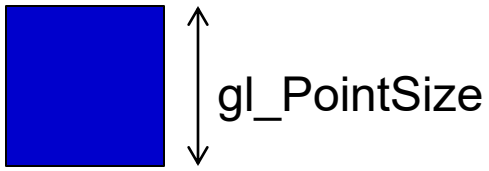
GL_Point - the point is one fragment.

- The vertex shader sets the point size:

```
gl_PointSize = 50;   //An example
gl_Position = mvpMatrix * position;
```

- The fragment shader processes each fragment of the square region:

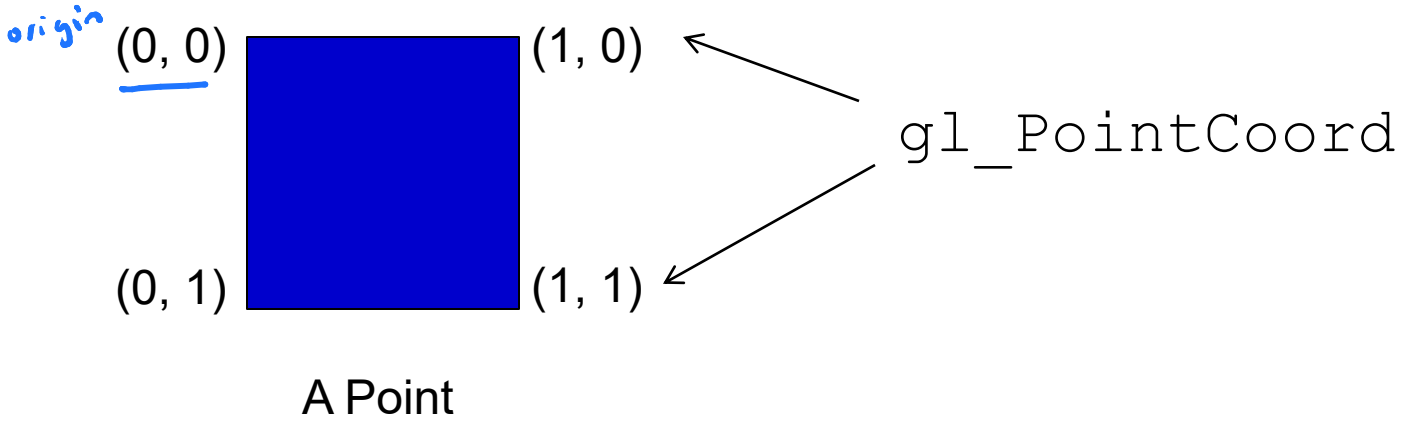
```
gl_FragColor = vec4(0, 0, 1, 1);
```



- Can we map a texture to this region?

Point Sprites

- The built-in fragment shader variable **`gl_PointCoord`** gives the coordinates of a fragment within a point.
- The *x* and *y* values range from 0 to 1, and hence can be directly used as texture coordinates.
- The default position of the origin is at the upper-left corner of the current point.

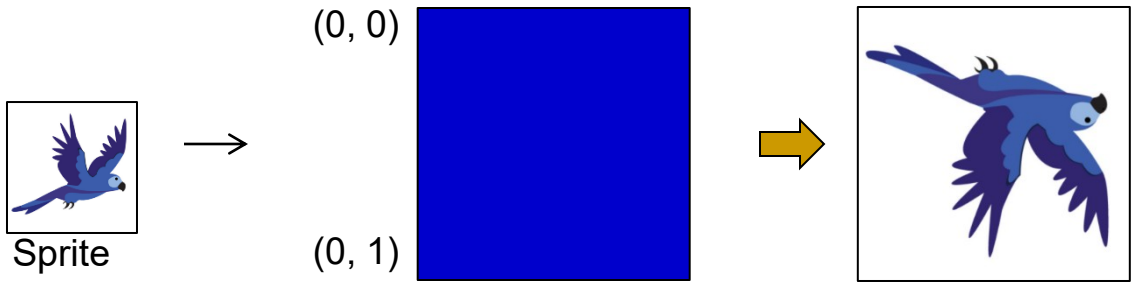


Point Sprites

Mapping a sprite texture

```
gl_FragColor = texture(spriteImg, gl_PointCoord);
```

- The *y*-values of `gl_PointCoord` increases from top to down, and causes a vertical flip of the mapped texture.



- The application can set the point's origin to bottom-left corner by calling

```
glPointParameteri(GL_POINT_SPRITE_COORD_ORIGIN, GL_LOWER_LEFT);
```

Point Sprites

Advantages:

- Requires only points to be generated. Texture coordinates for each sprite are not required to be stored.
- A point is always displayed as a screen aligned square region.

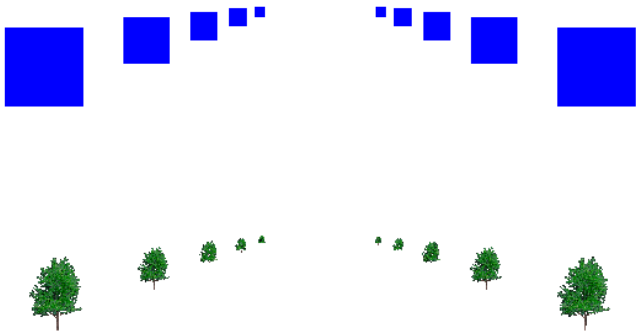
Limitations:

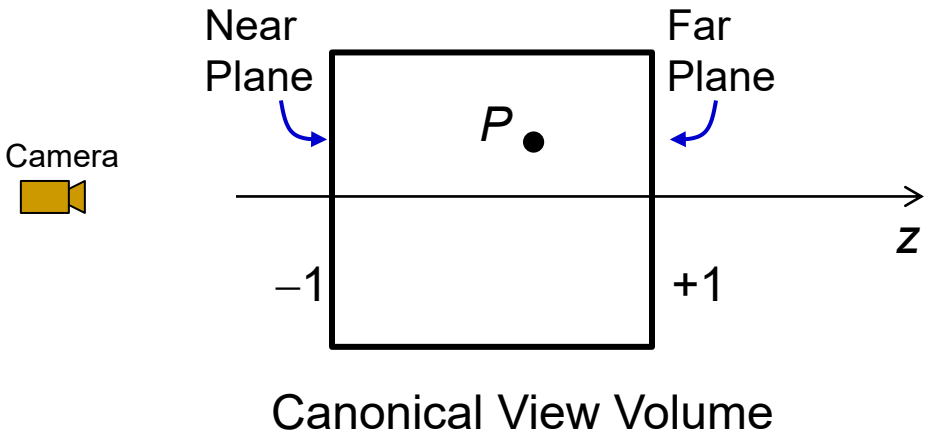
- Points are always square shaped.
- Point sprites can only be translated.
- Size specified in screen space.

Point Sprites

Computation of point size:

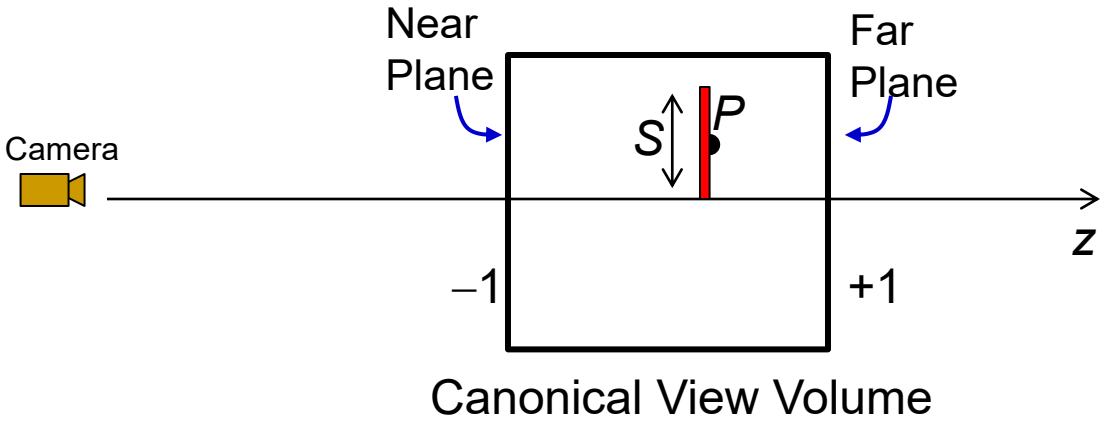
- It may be required to vary the size of a poit based on its distance from the camera to mimic perspective vision.
- We can use pseudo-depth value:
  - Clip coordinates: (x, y, z, w)
  - Pseudo-depth $d = z/w$. [-1, +1]





Canonical View Volume

Point Sprites



d	Size
+1	0
-1	*S*

Computation of Point Size:

- Let the clip coordiates of the point be (x, y, z, w)
- Let the required size of the sprite at its closest point to the camera (near plane) be *S*. At this point, *d* = -1.
- When the sprite is on the far plane, its size is 0. At this point, *d* = +1.
- Therefore, point size = $(1 - d)S/2 = (1 - (z/w))\ S/2$.

Billboards as Point Sprites

OpenGL-4 Implementation

Application

```
glEnable(GL_POINT_SPRITE);
glEnable(GL_PROGRAM_POINT_SIZE);
...
glDrawArrays(GL_POINTS, 0, n);
```

Vertex Shader

```
posnC = mvpMatrix * position;
gl_PointSize = (1.0 - posnC.z / posnC.w) * 64.0;
gl_Position = posnC;
```

Fragment Shader

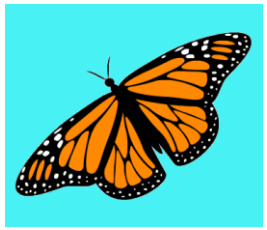
```
col = texture(texTree, gl_PointCoord);
if(col.a < 0.1) discard;
gl_FragColor = col;
```

Background Segmentation

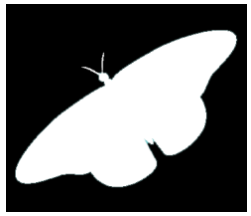
Point Sprites

Background Removal:

- RGBA Image: Use the alpha channel:



RGB



Alpha

```
col = texture(texTree, gl_PointCoord);
if(col.a < 0.1) discard;
gl_FragColor = col;
```

Fragment Shader

- RGB Image: Use background colour

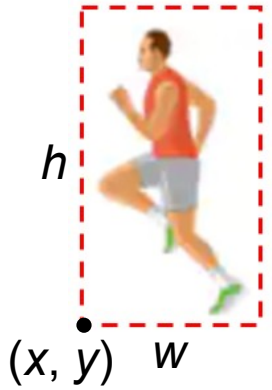
```
col = texture(texTree, gl_PointCoord);
if(col.r<0.05 && col.g>0.95 &&col.b<0.05) discard;
gl_FragColor = col;
```

Fragment Shader

Mapping a Non-square Texture

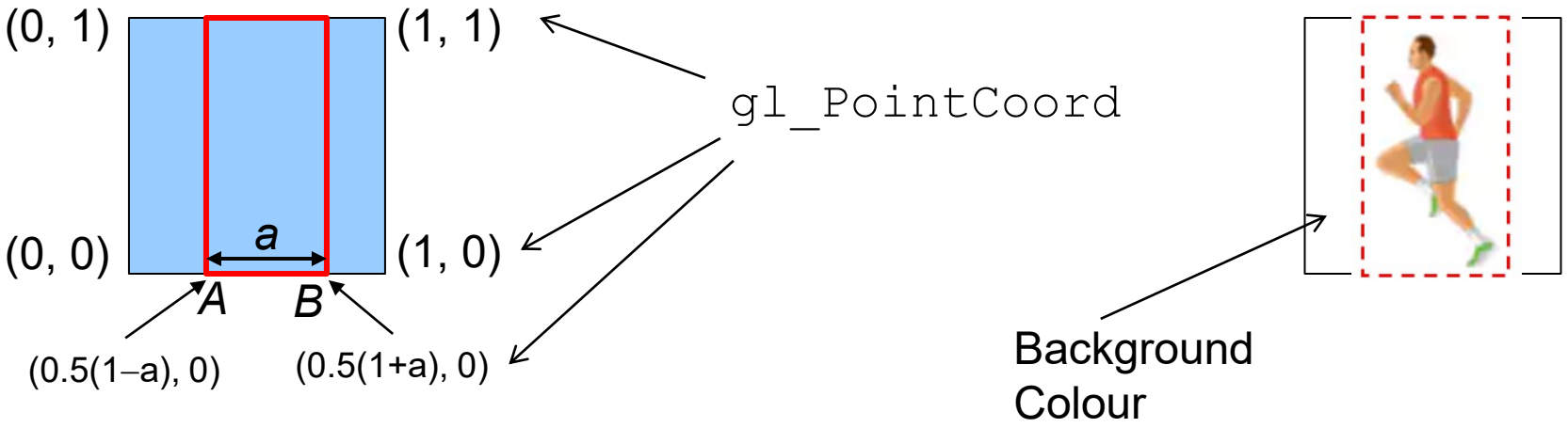
When the aspect ratio of a sprite is less than 1, we need to find a sub-region on the point with the same aspect ratio.

We cannot directly use gl_PointCoord as texture coordinates.
At point 'A', the texture coordinates must be (0, 0)
At point 'B' the texture coordinates must be (1, 0)

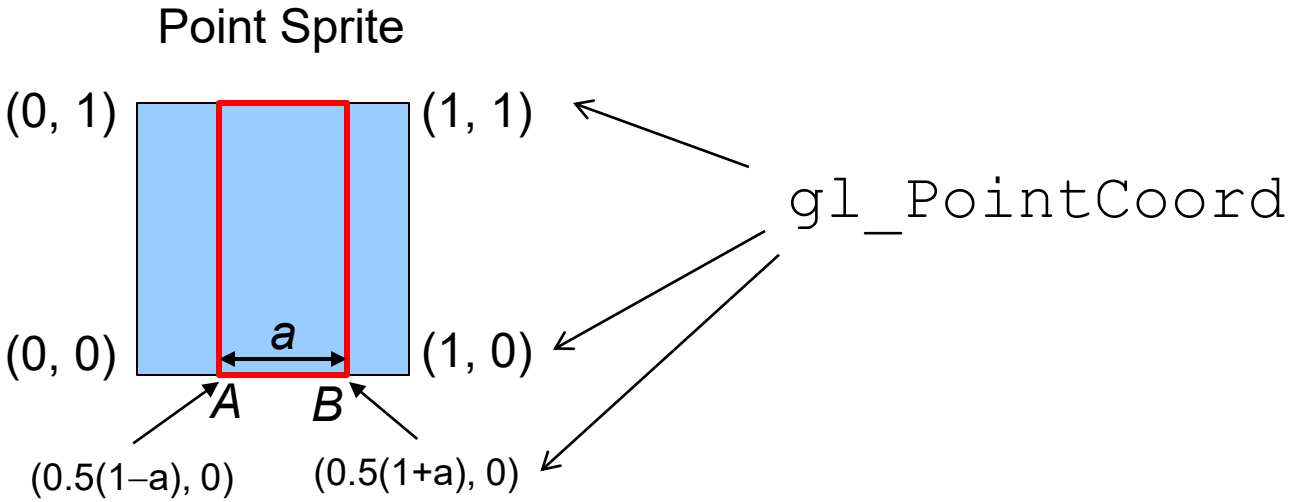


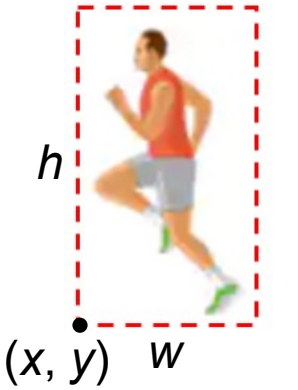
Aspect ratio
= a = w/h

Point Sprite



Mapping a Non-square Texture





Aspect ratio
= a = w/h

```
d1 = 0.5*(1-a);
d2 = 0.5*(1+a);
if(gl_PointCoord.x < d1 || gl_PointCoord.x > d2) col = vec4(1);
else
{
  tcoordx = (gl_PointCoord.x – d1) / (d2 – d1);
  tcoordy = gl_PointCoord.y;
  col = texture(texTree, vec2(tcoordx, tcoordy));
}
```

Horizontal Flipping

- A sprite sheet may not always contain all frames.
- The missing frames will need to be generated using operations such as horizontal flipping.

The construction of a walk sequence requires more sprites

- Solution:
  - Use an image editor to create additional frames from existing ones, or
  - Map the texture in the reverse direction along x-axis

    (tcoordx = 1 – tcoordx)

Sprites Using Geometry Shader

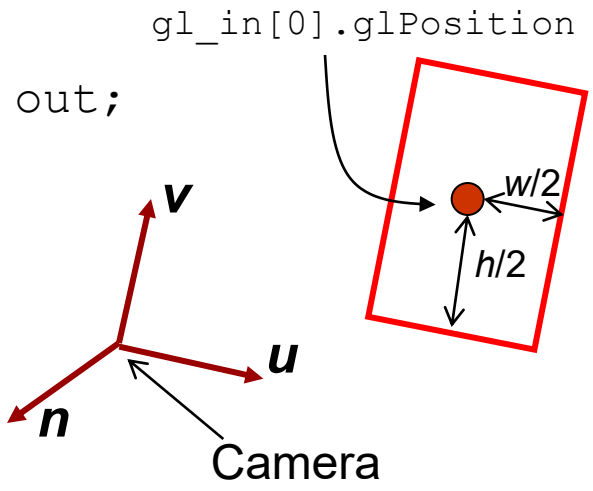
- A geometry shader gives more control over the way a sprite is constructed, transformed and displayed.
- Sample implementation:
  - Application generates points at the required positions
  - Vertex shader outputs positions in eye coordinates
  - Geometry shader constructs quads of the required size and shape in eye space and assigns texture coordinates.

Vertex Shader:

```
layout (location = 0) in vec4 position;
uniform mat4 viewMatrix;
void main()
{
   gl_Position = viewMatrix * position;
}
```

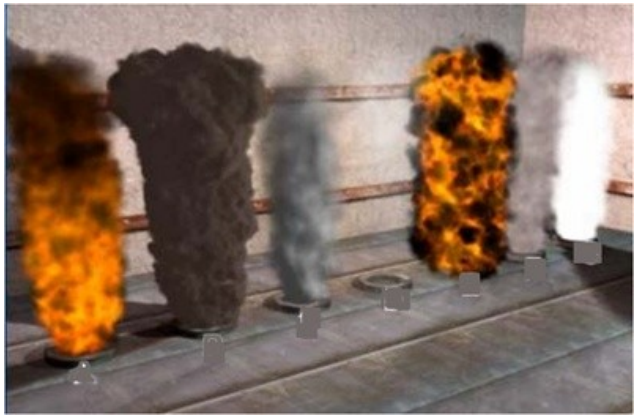
Sprites Using Geometry Shader

```
layout (points) in;
layout (triangle_strip, max_vertices = 4) out;
uniform float wid2;    //half width
uniform float hgt2;    //half height
uniform mat4 projMatrix;
out vec2 texCoord;
void main() {
  vec4 posn;
  posn = gl_in[0].glPosition + vec4(-wid2, -hgt2, 0, 0);
  gl_Position = projMatrix * posn;
  texCoord = vec2(0, 0);
  EmitVertex();
  posn = gl_in[0].glPosition + vec4(wid2, -hgt2, 0, 0);
  gl_Position = projMatrix * posn;
  texCoord = vec2(1, 0);
  EmitVertex();
   ...
```



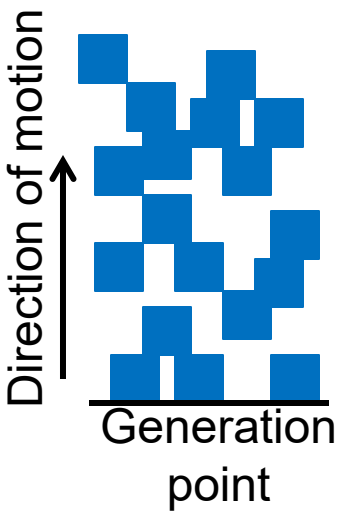
Particle Systems

- A system of particles whose state evolves over time using some physics based equations.
- Used for modelling fuzzy objects, irregular time varying shapes and visually complex natural phenomena
- Commonly used for rendering
  - Waterfalls, fountains
  - Fire, fireworks, explosions
  - Smoke, smoke trails, vapour trails



Particle Systems

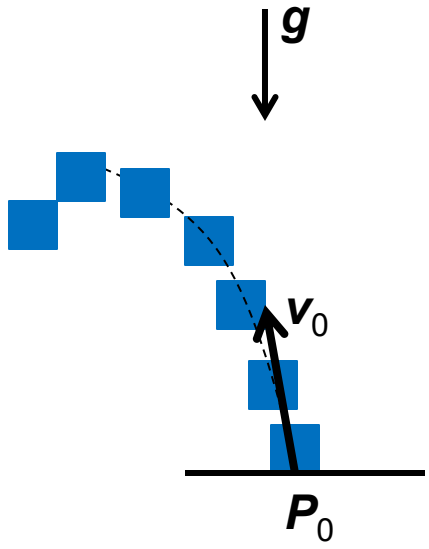
- Particle systems can be generated using point sprites.
- The main component of the system is a set of **update equations** that specify the changes in position and appearance of a particle with time.
- Each particle can have a set of attributes:
  - Initial position P_0
  - Initial velocity v_0 (and optionally acceleration a)
  - Lifetime T
  - Start time s_0
  - Alpha (for blending) α



Particle Systems

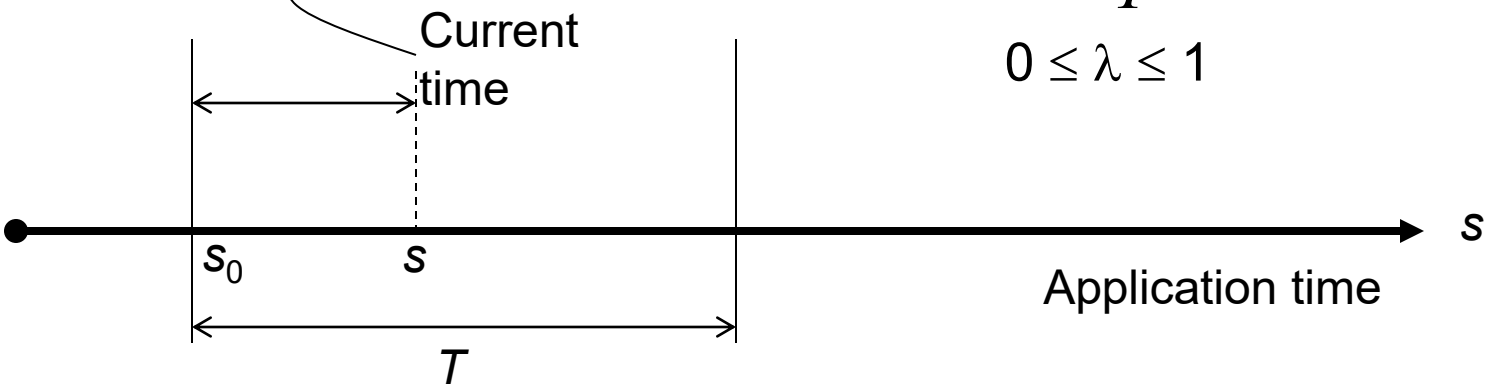
Position update:

$$P = P_0 + v_0 t + \left(\frac{1}{2}\right) a t^2$$



$t = s - s_0 \quad 0 \le t \le T \Rightarrow \lambda = \frac{t}{T}$

$0 \le \lambda \le 1$



For each particle, the values of P_0, v_0, s_0 can be stored in a VBO.

Particle Systems: Fire

Application creates 3 VBOs:

Number of particles

```
float vert[NPART], vel[NPART], startTime[NPART];

float t = 0;
for(int i = 0; i < NPART; i ++)
{
  vert[i] = glm::linearRand(-2.0f, 2.0f);
  vel[i] = glm::linearRand(1.5f, 2.5f);
  startTime[i] = t;
  t += 0.001;
}
```

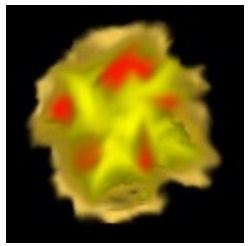
P_0

v_0

t_0

Particle Systems: Fire

Initial setup:



- The particles move only on a 2D plane.
- Each particle is a alpha-textured quad
- Each fragment of a newly rendered particle is blended with the colour at that position

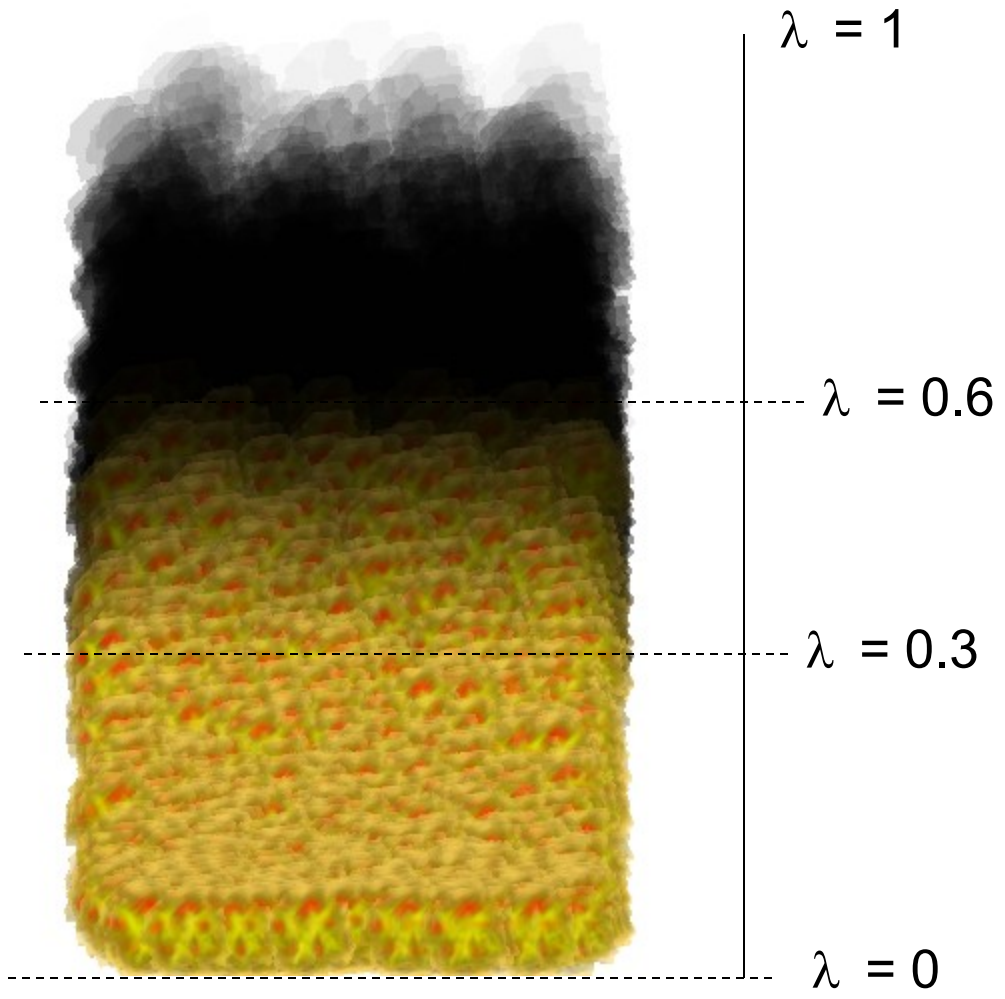
```
glDisable(GL_DEPTH_TEST);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_POINT_SPRITE);
glEnable(GL_PROGRAM_POINT_SIZE);
glPointParameteri(GL_POINT_SPRITE_COORD_ORIGIN, GL_LOWER_LEFT);
```

Particle Systems: Fire (Vertex Shader)

```
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 velocity;
layout (location = 2) in float startTime;
uniform mat4 mvpMatrix;
uniform float simt;     //Application time

flat out float lambda;
void main()  {
    vec4 pos;
    float finalTime = 3.0;
    float t = simt - startTime;
    if(t > 0 && t < finalTime)
    {
        pos =  vec4(position, 1.0);
        pos += vec4(velocity, 0) * t;
        gl_PointSize = 60.0;
        gl_Position = mvpMatrix * pos;
        lambda = t/finalTime;
    }
}
```

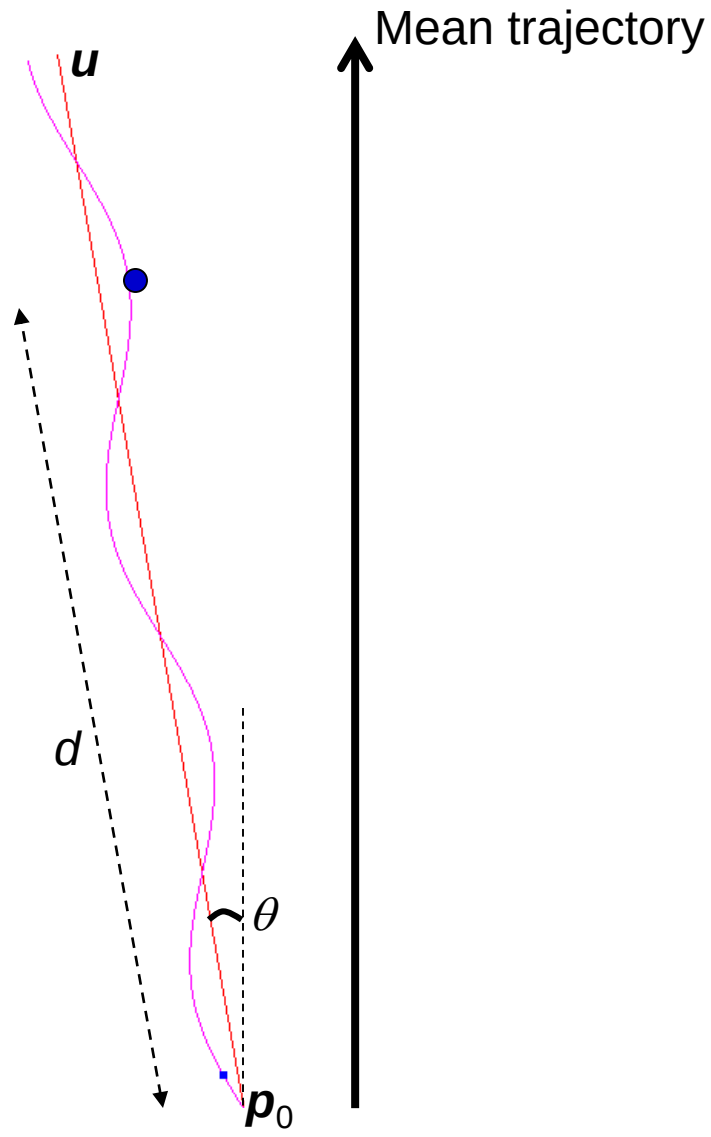
Particle Systems: Fire



Particle Systems: Implementation Issues

- Particle Queues – creating queues
  - Since particles have a fixed lifetime, many particle system implementations are based on a queue structure where new particles are generated and added to the queue as particles at the other end of the queue are removed.
  - In a shader based implementation, a static VBO is used. Here, instead of deleting and creating particles, particle positions are reset to the source point at the end of a cycle.
- Randomness – Adding randomness
  - Randomness of motion along the paths of particles can be modelled using low frequency sinusoidal variations.

Particle Motion: A Simple Model



- Parameters
  - Initial position $\boldsymbol{p}_0$
  - Course Deviation Angle (or direction vector $\boldsymbol{u}$)
  - Drift from main course modelled as a sine wave with magnitude $\boldsymbol{m}$, and angular frequency ω: $m \sin(\omega d)$, where d is the distance from source.
  - Velocity $\boldsymbol{v}$ of the particle. This is used to compute distance $d = v(t - t_0), \quad t > t_0$

Particle Motion: A Simple Model

- Parameter Ranges: Particles are assigned a random value within a pre-specified range for each of the parameters. E.g.:

7 VBOs

```
for(int i = 0; i < NPART; i ++)
{
        vert[i] =  glm::linearRand(-2.0f, 2.0f);
        vel[i] =   glm::linearRand(1.0f, 3.f);
        angle[i] = glm::linearRand(-10.0f, 10.0f);
        mag[i] =   glm::linearRand(0.1f, 1.0f);
        omeg[i] =  glm::linearRand(0.2f, 1.0f);
        startTime[i] = t;
        texindx[i] = glm::linearRand(0.0f, 3.9f);
        t += 0.1;
}
```

Particle Motion: Time

- The application time (simulation time) is continuously updated within the timer function.

```
void update(int value)
{
   simTime += 0.05;
   glutTimerFunc(50, update, 0);
   glutPostRedisplay();
}
```

```
uniform float simt;
```

Shader

- At time t_0, a particle is at position p_0, and for this particle, $d = 0$.

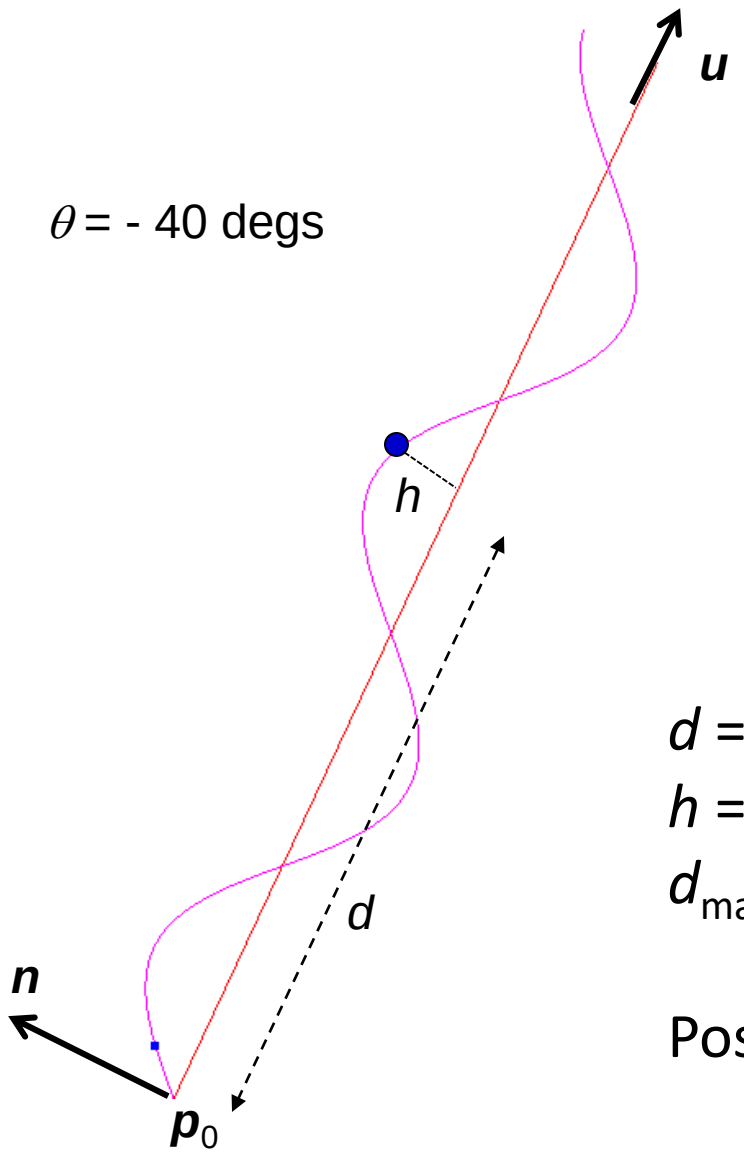
Particle Update: Vertex Shader

- The vertex shader receives the **initial** parameters corresponding to the current particle.
- It also receives the model-view-project matrix and the simulation (application) time.

```
layout (location = 0) in float px;
layout (location = 1) in float velocity;
layout (location = 2) in float startTime;
layout (location = 3) in float angle;
layout (location = 4) in float mag;
layout (location = 5) in float omeg;
layout (location = 6) in float tindx;

uniform mat4 mvpMatrix;
uniform float simt;
```

Computing Particle's Position



u : The main course vector of a particle.

2D: (-sinθ, cosθ)

3D: A unit vector

n: A vector orthogonal to ***u***.

2D: (-cosθ, -sinθ)

3D: ***view*** $\times$ ***u*** (***view*** = view vector)

d = mod ($v(t - t_0)$, d_{max})

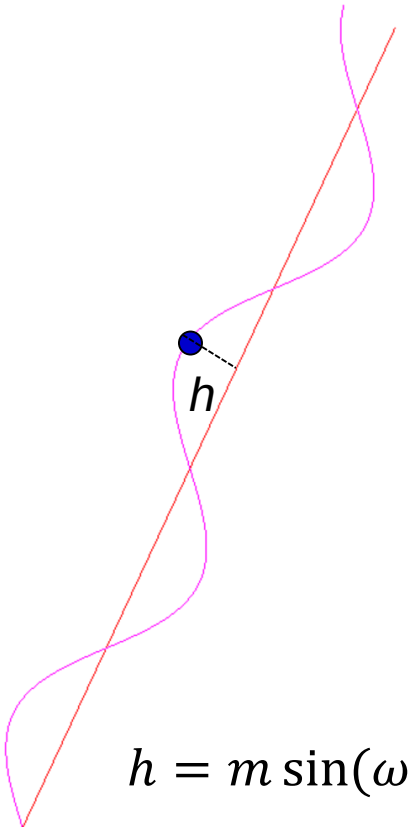
h = m sin(ωd)

d_{max} : The maximum distance from source.

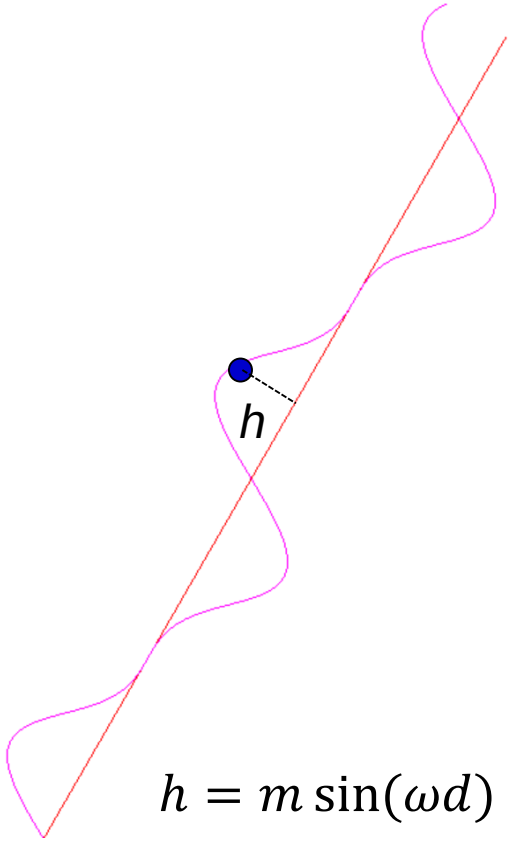
Position $\boldsymbol{p} = \boldsymbol{p_0} + d\,\boldsymbol{u} + h\,\boldsymbol{n}$

More Randomness

- The path of a particle may be further altered by adding more sinusoids with lower magnitude and higher frequency



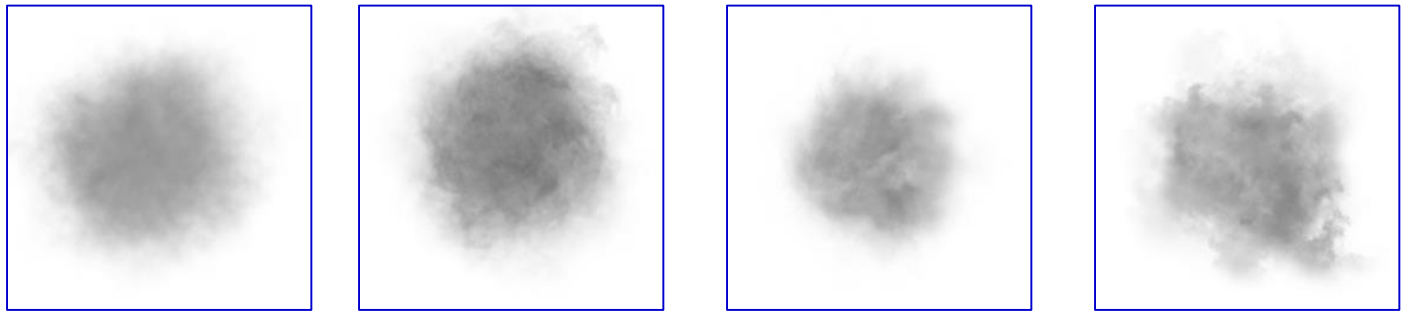
$$h = m\sin(\omega d)$$



$$h = m\sin(\omega d) + \left(\frac{m}{2}\right)\sin(2\omega d)$$

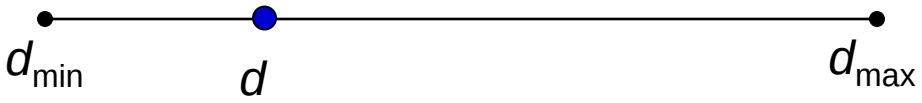
Texture - stored in VBO

- A set of similar textures may be used for texture mapping point sprites. A texture index is (randomly) assigned to each particle at the initialization stage.

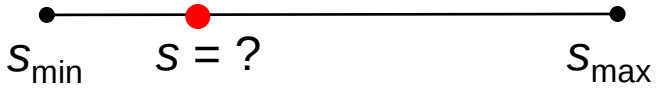


Smoke Textures

Linear Mapping

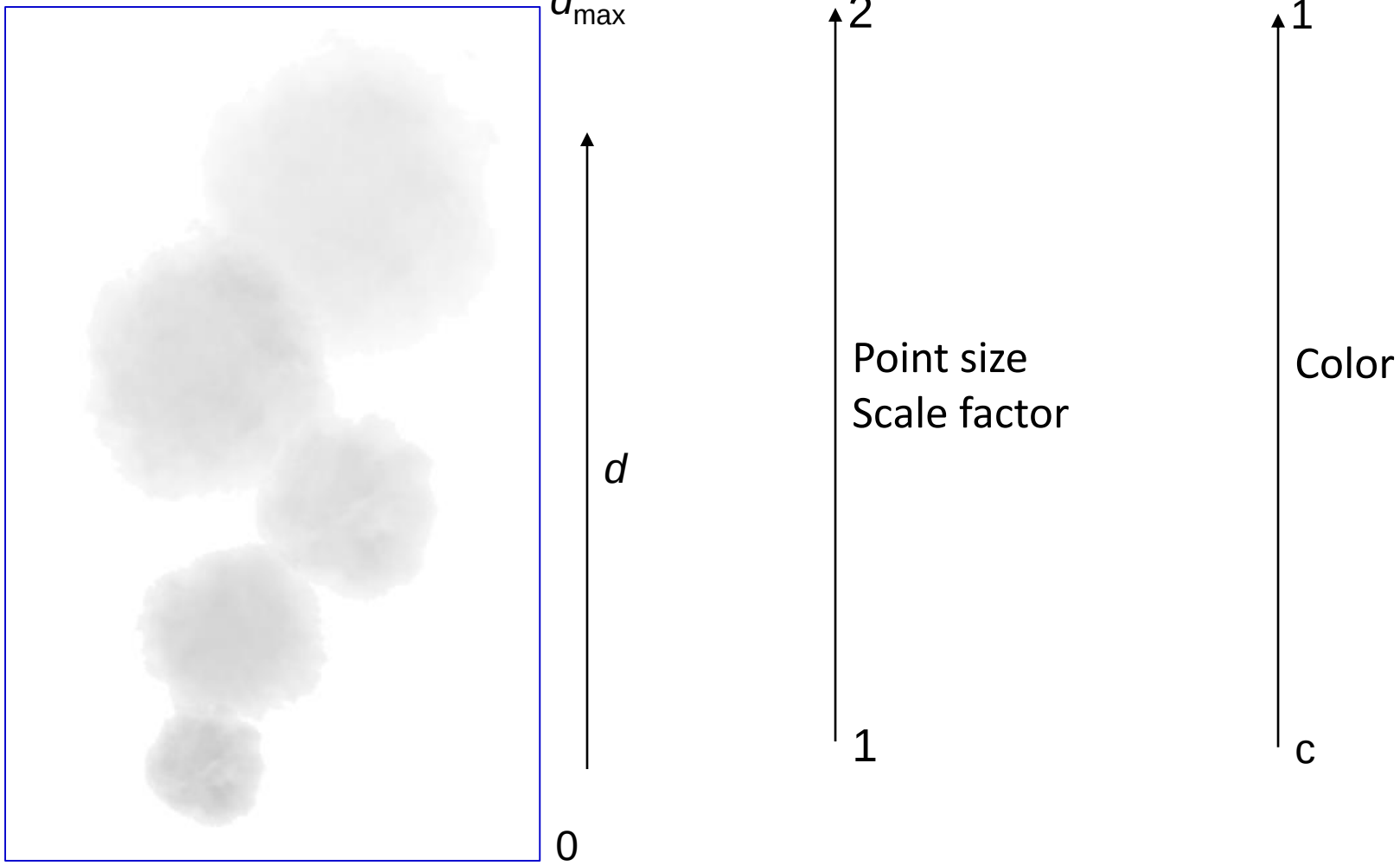


$$\frac{d - d_{min}}{d_{max} - d_{min}} = \frac{s - s_{min}}{s_{max} - s_{min}}$$



- We often require parametric variations based on distance ranges.
- E.g., As the particle's distance increases from d_{min} to d_{max}, its size should double (the scale factor for size should vary from 1 to 2)

Distance Based Scaling (Smoke)



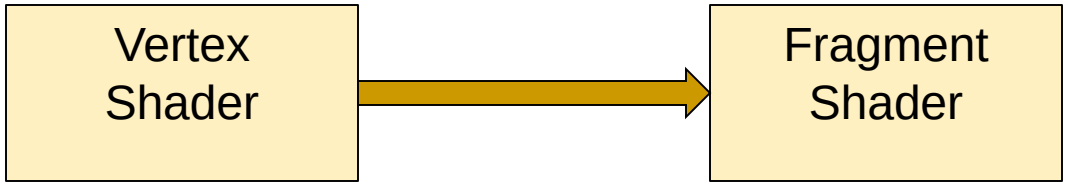
Particle Motion

- The required (fixed) number of particles are generated at the initialization step, and their initial attributes (position, velocity, angle, m, ω, $t0$, texIndex) are stored in VBO
- Distance d computed in vertex shader using velocity, application time and t_0.
- Current position computed based on distance
- Point size adjusted based on distance (vertex shader)
- Output colour adjusted based on distance (fragment)
- Blending enabled while drawing point sprites
- Each particle follows a pre-defined path

Smoke Particle System

Shader Implementations

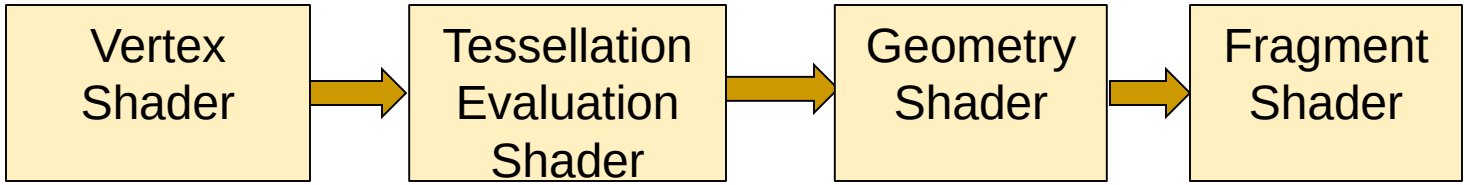
GL_POINTS



Compute position
Compute point size
Output dist, texIndex

Get colour from texture
Discard background fragments
Scale and output colour

GL_PATCHES



Compute position
Output dist, texIndex

Output gl_Position
Output dist, texIndex

Output Quad
Output dist, texIndex