Physics of a Mass Point & Basics of Textures

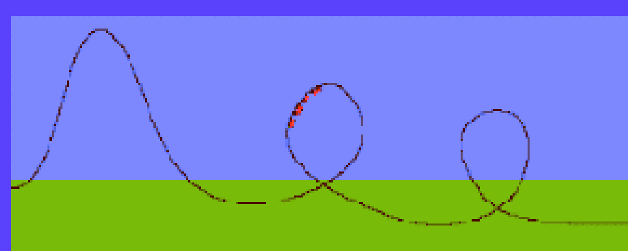
Point mass simulation
Basics of texture mapping in OpenGL

Chapter 8 in Watt

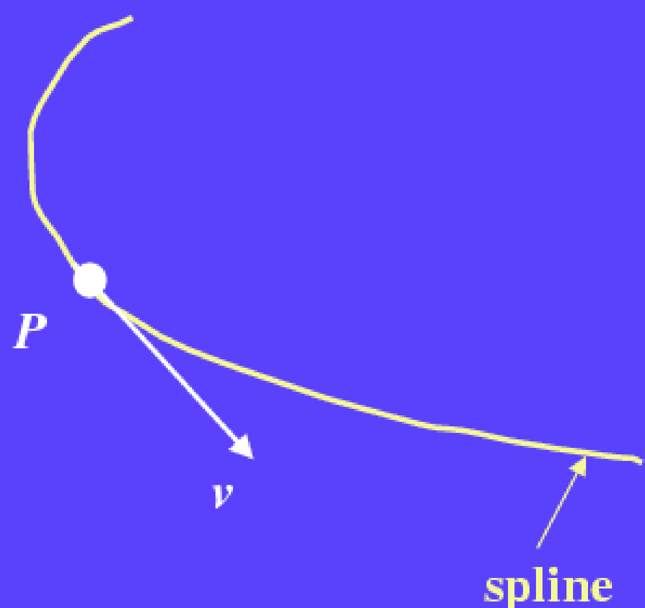
September 17 2002

Roller coaster

- Next programming assignment involves creating a 3D roller coaster animation
- We must model the 3D curve describing the roller coaster, but how?
- How to make the simulation obey the laws of gravity?



Back to the physics of the roller-coaster: mass point moving on a spline



frictionless model, with gravity

- Velocity vector always points in the tangential direction of the curve

Mass point on a spline (contd.)
frictionless model, with gravity

- Our assumption is : no friction among the point and the spline
- Use the conservation of energy law to get the current velocity
- $W_{kin} + W_{pot} = const = m * g * h_{max}$
- h_{max} reached when $|v|=0$
- W_{kin} = kinetic energy = $1/2 * m * |v|^2$
- W_{pot} = potential energy = $m * g * h$
- h = the current z-coordinate of the mass point
- g = acceleration of gravity = 9.81 ms^{-2}
- m = mass of the mass point

Mass point on a spline (contd.)
frictionless model, with gravity

- Given current h, we can always compute the corresponding |v|:

$$|v| = \sqrt{2g(h_{\max} - h)}$$

Mass point motion*

- Assume we know the initial position of a mass point, and velocity $v=v(t)$
- Velocity is a 3-dim vector
- Problem: compute the position of the point at an arbitrary time t_1
- Has to integrate velocity over time:

$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} v(t)\,dt$$

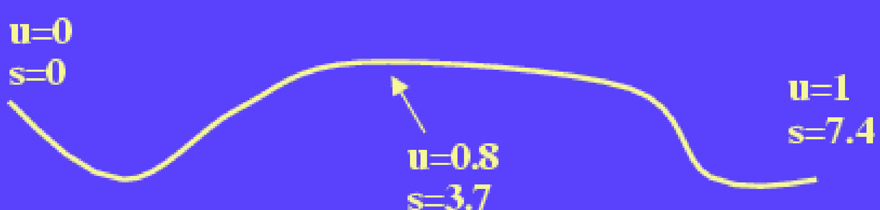
- x, v are vectors

Mass point motion (contd.)*

- Usually, cannot compute the integral symbolically
- Numerical integration necessary
- Standard numerical integration routines can be used (i.e. Simpson, Trapezoid, etc.)
- Integrate each of the coordinates x,y,z separately
- This is a general approach
  - For motion on a spline, use arclength parameterization approach instead

Arclength Parametrization

- There are an infinite number of parameterizations of a given curve. Slow, fast, speed continuous or discontinuous, clockwise (CW) or CCW…
- A special one: arc-length-parameterization: u=s is arc length. We care about these for animation.



need control over velocity along the curve for animation

- Problem: parameterizations usually aren't arc-length
- How to transform parameterization to an arc-length parameterization?

Arclength Parametrization (contd.)

- Assume a general parameterization p=p(u)
- $p(u) = [x(u), y(u), z(u)]^T$
- arclength parameter s=s(u) is the distance from p(0) to p(u) along the curve
- Distance increases monotonically, hence s=s(u) is a monotonically increasing function
- It follows from Pitagora's law that

$$s(u) = \int_0^u \sqrt{x'(v)^2 + y'(v)^2 + z'(v)^2}\, dv$$

Arclength parameter *s*

- The integral for s(u) usually cannot be evaluated analytically, not even for cubic splines (simple polynomials)
- Has to evaluate the integral numerically
- Simpson's integration rule (next slide)
- Piecewise polynomial definition of the spline means we have to break the integral over individual spline pieces
- For a fixed spline, can pre-compute function s=s(u) for certain values of u and store it into an array
- For the next slides, we will assume we have a routine, which computes s(u), given a value of u

Simpson integration rule

$$\int_a^b f(x)dx = \sum_{k=1}^{(n-1)/2} \frac{h}{3}[f(x_{2k-1}) + 4f(x_{2k}) + f(x_{2k+1})] + O(h^5)$$

- $a = x_1$, $b=x_n$, $h=(b-a)/(n-1)$
- $h = x_{2k+1}-x_{2k} = x_{2k}-x_{2k-1}$ = independent of k
- n > 3 corresponds to the number of intervals
- formula exact for a cubic polynomial
- n MUST be odd
- Must be able to evaluate the function at the points $x_{2k-1}, x_{2k}, x_{2k+1}$
- Alternative to Simpson: Trapeziod rule
  - Less accurate: Error is O(h3)
  - Simpler to compute than Simpson

Inverse u=u(s)

- Inverse problem:
  Given arclength s, determine the original parameter u
- Since s=s(u) is monotonically increasing, so is u=u(s)
- Useful (necessary) for animating motion along the curve
- Since u=u(t) can only be computed numerically, there is no exact formula for u=u(s)

Computing inverse u=u(s)

- Given arclength s, we can use bisection to determine the corresponding u
- Can compute (using Simpson's rule) the function s=s(u) in the forward direction

Computing inverse u=u(s)

- Must have initial guess for the interval containing u

```
Bisection(umin,umax,s)
/* umin = min value of u
   umax = max value of u; umin <= u <= umax
    s = target value */
    Forever // but not really forever
    {
        u = (umin + umax) / 2; // u = candidate for solution
        If |s(u)-s| < epsilon
            Return u;
        If s(u) > s  // u too big, jump into left interval
            umax = u;
        Else  // t too small, jump into right interval
            umin = u;
    }
```

Simulating mass point on a spline

- Assume we know the size of the current velocity vector |v| of a mass particle on the spline at a given moment in time t
  - Can obtain this using the laws of physics, as shown before
- Notation:
  - u = original parameterization
  - t = time
  - s = natural parameterization (i.e. arclength parameterization)
- We keep current u, t and s in three separate variables
- How to compute the next position of the particle?

Simulating mass point on a spline

- Time step Δt
- We have: $\Delta s = |v| * \Delta t$ and $s = s + \Delta s$.
- We want the new value of u, so that can compute new point location
- Therefore:
  We know s, need to determine u
  Here we use the bisection routine to compute u=u(s).

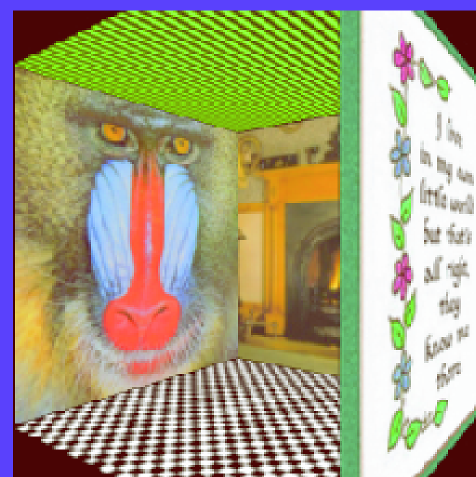
Mass point simulation

- Assume we have a 32-piece spline, with a general parameterization of $u \in [0,31]$

```
MassPoint(tmax) // tmax = final time
/* assume initially, we have t=0 and point is located at
u=0 */
    u = 0;
    s = 0;
    t = 0;
    While t < tmax
    {
        Assert u < 31;  // if not, end of spline reached
        Determine current velocity |v| using physics;
        s = s + |v| * Δt; // compute new arclength
        u = Bisection(u,u + delta,s); // solve for t
        p = p(u); // p = new mass point location
        Do some stuff with p, i.e. render point location, etc.
        t = t + Δt; // proceed to next time step
    }
```

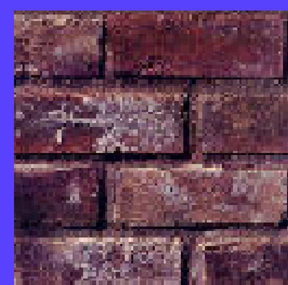
Texture Mapping

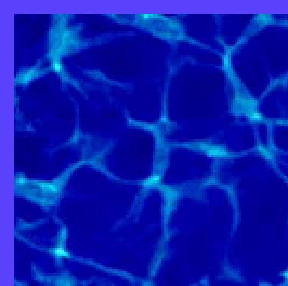
- A way of adding surface details
- Two ways can achieve the goal:
  - Model the surface with more polygons
    - » Slows down rendering speed
    - » Hard to model fine features
  - Map a texture to the surface
    - » This lecture
    - » Image complexity does not affect complexity of processing



The texture

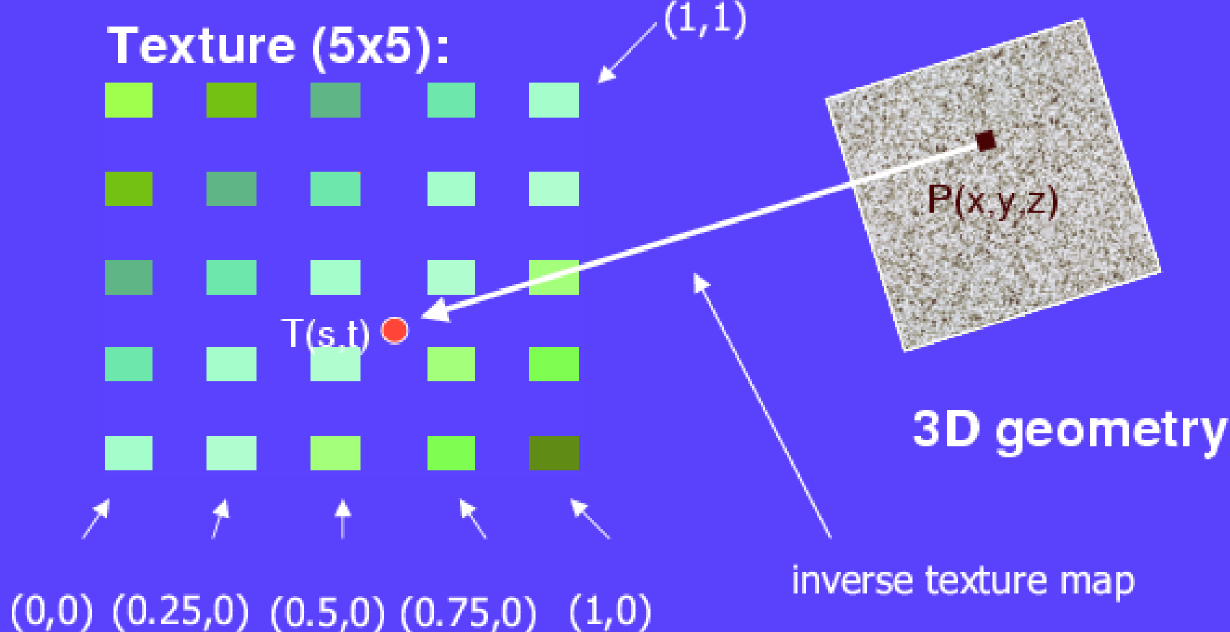
- Texture is a bitmap image
  - Can use libpicio library to load image into memory
  - Or can create image yourself within the program
- 2D array: texture[height][width][4]
- Pixels of the texture called *texels*
- Texel coordinates (s,t) scaled to [0,1] range





Texture Value Lookup

- For given texture coordinates (s,t), we can find a unique image value, corresponding to the texture image at that location

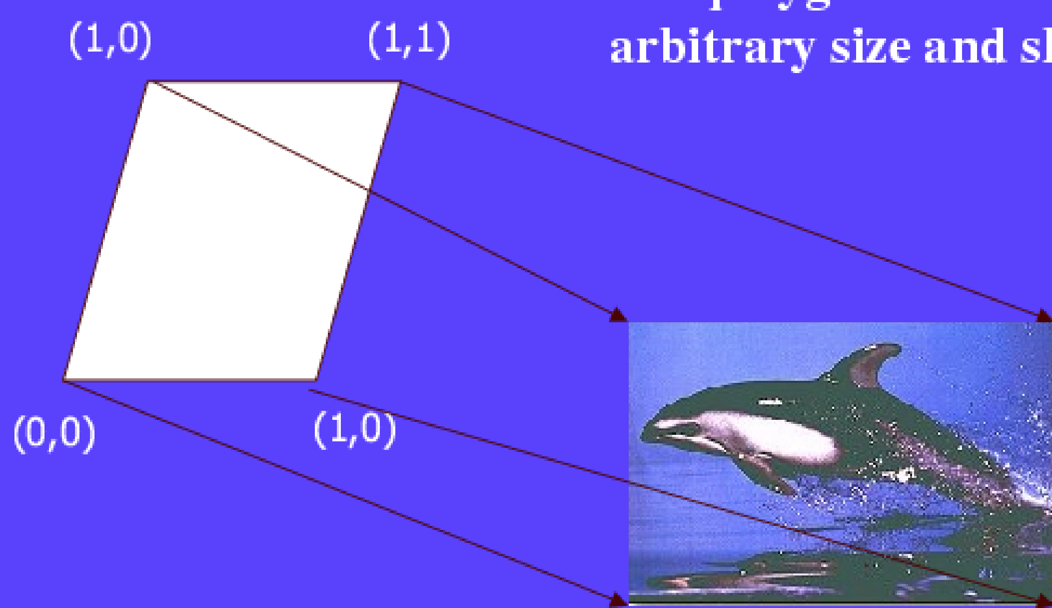


Interpolating colors

- Some (s,t) coordinates not directly at pixel in the texture, but in between
- Minifaction, magnification
- Solutions:
  - Nearest neighbor
    - Use the nearest neighbor to determine color
    - Faster, but worse quality
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  - Linear interpolation
    - Incorporate colors of several neighbors to determine color
    - Slower, better quality
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)

Map textures to surfaces

The polygon can have arbitrary size and shape



Color blending

- Final pixel color = f (texture color, object color)
- How to determine the color of the final pixel?
  - GL_MODULATE – multiply texture and object color
  - GL_BLEND – linear combination of texture and object color
  - GL_REPLACE – use texture color to replace object color
- Example:
  - glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

What happens if texture coordinates outside [0,1] ?

- Two choices:
  - Repeat pattern (GL_REPEAT)
  - Clamp to maximum/minimum value (GL_CLAMP)

- Example:
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)

Texture mapping in OpenGL

- In init():
  - Specify texture
    - » Read image from file into an array in memory or generate the image using the program
  - Specify texture mapping parameters
    - » Wrapping, filtering, etc.
  - Define (activate) the texture
- In display():
  - Enable GL texture mapping
  - Draw objects: Assign texture coordinates to vertices
  - Disable GL texture mapping

Specifying texture mapping parameters

- Use glTexParameteri
- Example:

```
// texture wrapping on
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT); // repeat pattern in s texture coordinate
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_REPEAT); // repeat pattern in t texture coordinate

// use nearest neighbor for both minification and magnification
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
```

Defining (activating) texture

- Do once in init() to set up initial pattern
- To use another texture, make further calls in display() to glTexImage2D, specifying another image
  - But this is slow: use Texture Objects itself
- The dimensions of texture images **must be powers of 2**
  - if not, rescale image or pad with zeros
- glTexImage2D(Glenum target, Glint level,
  Glint internalFormat, int width, int height, Glint border,
  Glenum format, Glenum type, Glvoid* img)
- Example:
  - glTexImage2D(GL_TEXTURE_2D, 0,
    GL_RGBA, 256, 256, 0,
    GL_RGBA, GL_UNSIGNED_BYTE, pointerToImage)

Enable/disable texture mode

- Can do in init() or successively in display()
- glEnable(GL_TEXTURE_2D)
- glDisable(GL_TEXTURE_2D)

- Successively enable/disable texture mode to switch between drawing textured/non-textured polygons
- Changing textures:
  - Only one texture active at any given time
  - make another call to glTexImage2D to make another pattern active

The drawing itself

- Use GLTexCoord2f(s,t) to specify texture coordinates
- State machine: Texture coordinates remain valid until you change them or exit texture mode via glDisable (GL_TEXTURE_2D)
- Example:

```
glEnable(GL_TEXTURE_2D)
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
glTexCoord2f(0.0,1.0); glVertex3f(-2.0,1.0,0.0);
glTexCoord2f(1.0,0.0); glVertex3f(0.0,1.0,0.0);
glTexCoord2f(1.0,1.0); glVertex3f(0.0,-1.0,0.0);
...
glEnd();
glDisable(GL_TEXTURE_2D)
```

Everything together

```
void init(void):
{
…
put image into 2D memory array; // can use libpicio library

// create placeholder for texture
glGenTextures(1, &texName); // GLUint texName
glBindTexture(GL_TEXTURE_2D,texName);

// specify texture parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT); // repeat pattern in s
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT); // repeat pattern in t

// use nearest neighbor for both minification and magnification
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// make the pattern at location pointerToImage the active pattern
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, pointerToImage)

…
}
```

Everything together (contd.)

```
void display(void):
{
…
// no blending, use texture color directly
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, GL_REPLACE);
// turn on texture mode
glEnable(GL_TEXTURE_2D);

glBegin(GL_QUADS); // draw a quad
glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
glTexCoord2f(0.0,1.0); glVertex3f(-2.0,1.0,0.0);
glTexCoord2f(1.0,0.0); glVertex3f(0.0,1.0,0.0);
glTexCoord2f(1.0,1.0); glVertex3f(0.0,-1.0,0.0);
…
glEnd();

// turn off texture mode
glDisable(GL_TEXTURE_2D);

// draw some non-texture mapped objects
…
// switch back to texture mode, etc.
…
}
```