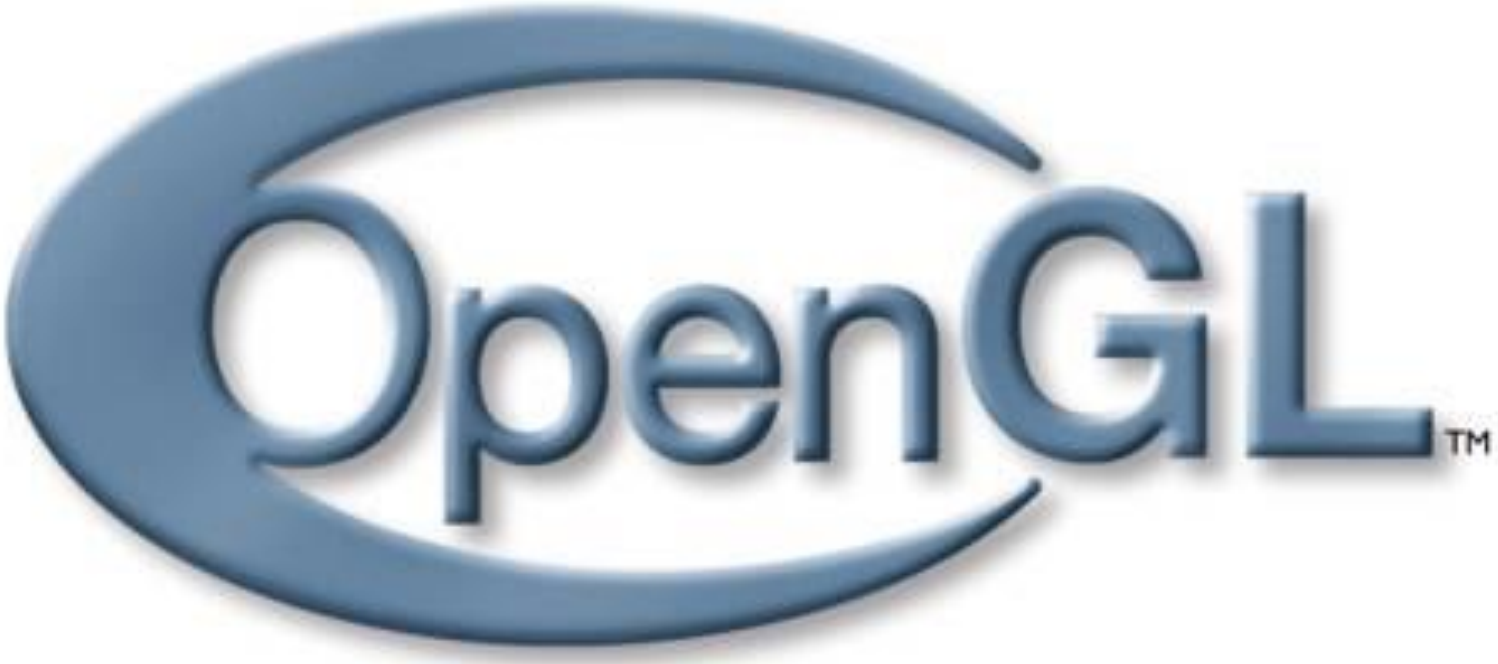


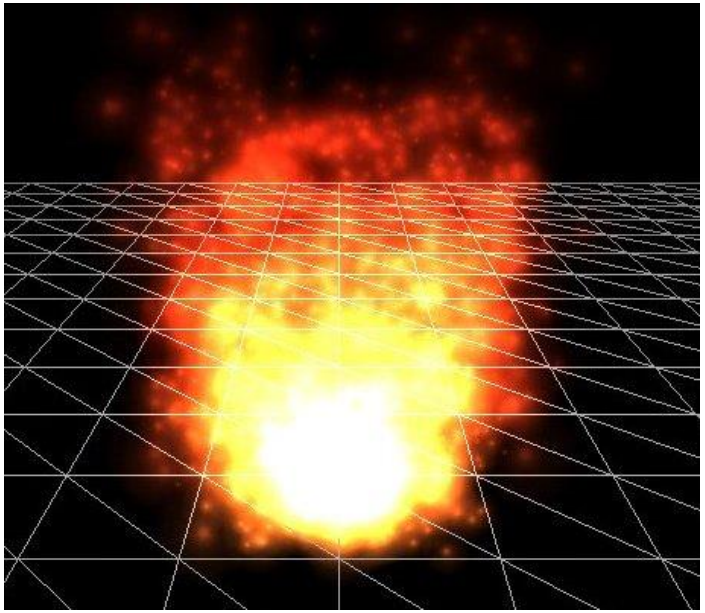
part 11

Martin Samuelčík
http://www.sccg.sk/~samuelcik
Room I4

Particle system

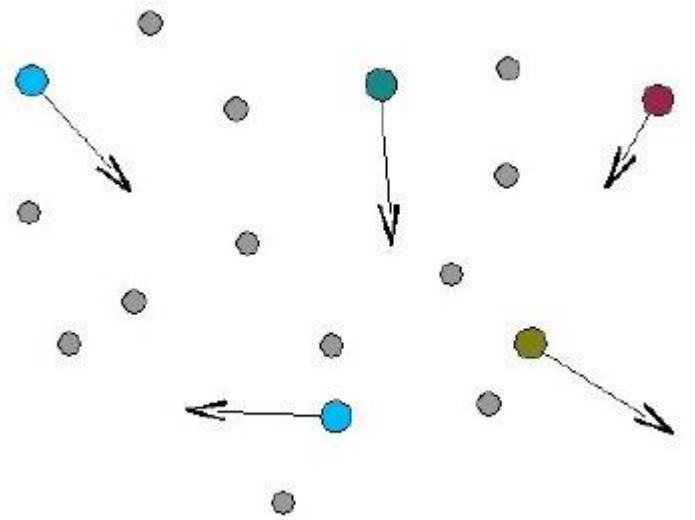
- System that consists of lots of small elements – particles
- Particles usually represented as points in space and changing in time
- Particle parameters – position, velocity, color, energy, size
- External forces on particles – gravity, wind, neighbor particles
- Simulating fire, explosion, rain, smoke, waterfall, stars, clouds, ……

Particle system

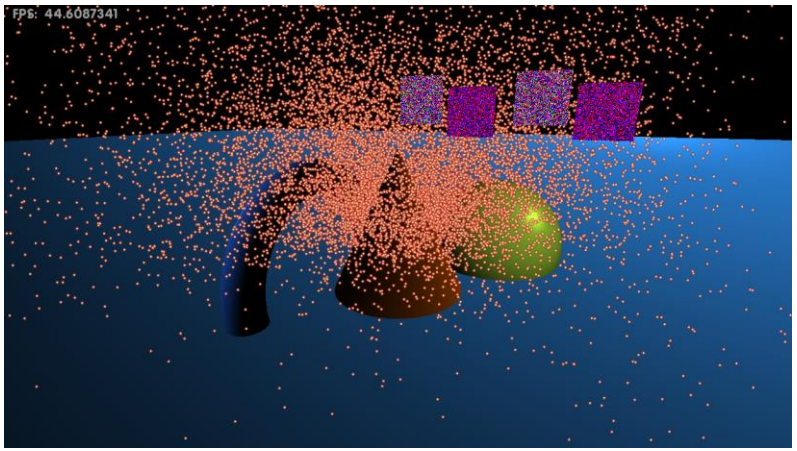


System loop

- Simulation stage
  - Update of particles



- Rendering stage
  - Visualization of particles



Simulation stage

- Emitter – shape that defines area or volume where new particles will be generated
- Creation of new particles inside emitter and setting default particle parameters
- Updating parameters (position, velocity, …) of each particle based on particle parameters and external forces
- Checking for collision with objects and particles during update
- Removing particles that are too old or fulfill other termination criteria (out of bounds)

Rendering stage

- Each particle is rendered independently
- Rendering as point or as quad
- Rendering as billboards – always facing camera
- Adding transparency and textures
- Using LODs for near-far particles
- Some simulations – creating surface from particles and rendering meshes

Particle rendering

- Immediate mode
  - Particles are stored in CPU memory
  - CPU update of positions
  - Use GL_POINTS or GL_QUADS for rendering each particle
- VBO
  - Particles positions are stored in GPU memory
  - CPU update of mapped positions
  - Use GL_POINTS or GL_QUADS

Point parameters

- For GL_POINTS rendering mode
- Setting additional parameters for points
- Extensions GL_ARB_point_parameters, GL_ARB_point_sprite
- **void glPointParameterfARB (enum pname, float param), void PointParameterfvARB (enum pname, const float *params)**
  - pname - POINT_SIZE_MIN_ARB, POINT_SIZE_MAX_ARB, POINT_FADE_THRESHOLD_SIZE_ARB, POINT_DISTANCE_ATTENUATION_ARB
  - Setting minimal and maximal size of point in pixels, how the size is attenuated by the distance of point from camera
- Mapping textures on rendered points

New approaches

- Geometry shader
  - Only points are rendered
  - In geometry shader, from each point, one textured quad is created
- Instance rendering
  - Only one point or quad is rendered
  - OpenGL creates as many copies, instances of point or quad as necessary
  - Positions of points or quads are sent to vertex shader as array of uniforms

Example

Inspired by http://www.codesampler.com/oglsrc/oglsrc_6.htm

```
struct Particle
{
    float position[3];
    float velocity[3];
    float color[4];
    float life;
};
Particle g_particle_system[1000];


float getRandomMinMax( float fMin, float fMax )
{
    float fRandNum = (float)rand () / RAND_MAX;
    return fMin + (fMax - fMin) * fRandNum;
}

Void  getRandomVector(float* vector)
{
    vector[2] = getRandomMinMax( 0.0f, 1.0f );
    float radius = (float)sqrt(1 - vector[2] * vector[2]);
    float t = getRandomMinMax( -PI, PI );
    vector[0] = (float)cosf(t) * radius;
    vector[1] = (float)sinf(t) * radius;
    return vector;
}
```

```
void initParticles()
{
    for( int i = 0; i < 1000; i++ )
    {
        // emitter of all particles is point [0,0,0]
        g_particle_system[i].position[0] = 0;
        g_particle_system[i].position[1] = 0;
        g_particle_system[i].position[2] = 0;
        g_particle_system[i].velocity = getRandomVector();
        g_particle_system[i].color[0] = getRandomMinMax( 0.0f, 1.0f );
        g_particle_system[i].color[1] = getRandomMinMax( 0.0f, 1.0f );
        g_particle_system[i].color[2] = getRandomMinMax( 0.0f, 1.0f );
        g_particle_system[i].life = getRandomMinMax( 5.0f, 20.0f );
    }
}
```

Example

```
void updateParticles()
{
   static double dLastFrameTime = timeGetTime();
   double dCurrenFrameTime = timeGetTime();
   double dElpasedFrameTime = (float)((dCurrenFrameTime - dLastFrameTime) * 0.001);
   dLastFrameTime = dCurrenFrameTime;
   for( int i = 0; i < 1000; i++ )
   {
      g_particle_system[i].position[0] += (float)dElpasedFrameTime * g_particle_system[i].velocity[0];
      g_particle_system[i].position[1] += (float)dElpasedFrameTime * g_particle_system[i].velocity[1];
      g_particle_system[i].position[2] += (float)dElpasedFrameTime * g_particle_system[i].velocity[2];
      g_particle_system[i].life -= dElpasedFrameTime;
      if (g_particle_system[i].life < 0)
      {
         g_particle_system[i].position[0] = 0;
         g_particle_system[i].position[1] = 0;
         g_particle_system[i].position[2] = 0;
         g_particle_system[i].velocity = getRandomVector();
         g_particle_system[i].color[0] = getRandomMinMax( 0.0f, 1.0f );
         g_particle_system[i].color[1] = getRandomMinMax( 0.0f, 1.0f );
         g_particle_system[i].color[2] = getRandomMinMax( 0.0f, 1.0f );
         g_particle_system[i].life = getRandomMinMax( 5.0f, 20.0f );
      }
   }
}
```

Example

```
void renderParticles()
{
    glEnable( GL_BLEND );
    glBlendFunc( GL_SRC_ALPHA, GL_ONE );
    glDepthMask(GL_FALSE);
    float quadratic[] =  { 1.0f, 0.0f, 0.01f };
    glPointParameterfvARB( GL_POINT_DISTANCE_ATTENUATION_ARB, quadratic );
    glPointSize(100);
    glPointParameterfARB( GL_POINT_FADE_THRESHOLD_SIZE_ARB, 60.0f );
    glPointParameterfARB( GL_POINT_SIZE_MIN_ARB, 1.0f );
    glPointParameterfARB( GL_POINT_SIZE_MAX_ARB, 100.0f );

    glBindTexture( GL_TEXTURE_2D, g_id_point_sprite_texture );
    glTexEnvf( GL_POINT_SPRITE_ARB, GL_COORD_REPLACE_ARB, GL_TRUE );
    glEnable( GL_POINT_SPRITE_ARB );
    glBegin( GL_POINTS );
    for( int i = 0; i < 1000; i++ )
    {
        glColor3fv(g_particle_system[i].color);
        glVertex3fv(g_particle_system[i].position);
    }
    glEnd();
    glDisable( GL_POINT_SPRITE_ARB );
    glDepthMask(GL_TRUE);
}
```

Selection

- Mechanism for identifying objects inside viewing volume - viewing volume can be changed during selection
- Computing intersection of box or pyramid with objects
- Picked objects are returned as previously defined names
- It’s different as normal render mode, values in framebuffer are not changed

Selection pipeline

- Define the viewing volume for selection using classic transformation functionality
- Init name stack
- Define array for returned hits
- Set selection mode:
- Render objects using drawing commands, for each object assign name (identifier) using name stack commands for each primitive of interest
- Exit selection mode and process hit records stored in defined array

Selection buffer & mode

- **void glSelectBuffer(GLsizei size, GLuint *buffer)**
  - Defining buffer where all hits will be recorded
  - Hit = rendered object was inside current viewing volume
  - size – size of array in bytes
  - buffer – already allocated array for hits
- **GLint glRenderMode(GLenum mode)**
  - Switching between modes
  - mode - GL_RENDER (the default), GL_SELECT, or GL_FEEDBACK
  - Function returns number of hits
  - When switching from selection to render mode -> selection buffer is filled

Name stack

- Names – non-negative integers
- **void glInitNames(void)**
  - Clears stack – no names are in stack after clearing
- **void glPushName(GLuint name)**
  - Adds name at top of stack
- **void glPopName(void)**
  - Removes name from top of stack
- **void glLoadName(GLuint name)**
  - Changes name at top of stack
- Name at the top of stack is added to currently drawing object in selection mode

Hit record

- In the selection array, set of consecutive hits is stored
- Each hit record consists of
  - The number of names on the name stack when the hit occurred
  - Minimum and maximum z values in window coordinates for all vertices of the primitives that are inside the viewing volume
  - The contents of the name stack at the time of the hit, with the bottommost element first

Selection example

```
void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   // render scene normally, using perspective camera
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluPerspective (40.0, 4.0/3.0, 0.01, 100.0);
   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();
   gluLookAt (7.5, 7.5, 12.5, 0, 0, 0, 0.0, 1.0, 0.0);
   drawWorldScene (false);

   // init selection buffer
   GLuint selectBuf[BUFSIZE];
   glSelectBuffer (BUFSIZE, selectBuf);
   // render scene with ortho projection in selection mode
   // this will find all objects inside box <0,5>x<0,5>x<0,10>
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glOrtho (0.0, 5.0, 0.0, 5.0, 0.0, 10.0);
   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();
   glRenderMode (GL_SELECT);
   drawWorldScene (true);
   glFlush ();
   // return to normal render mode and write result of selection
   GLint hits = glRenderMode (GL_RENDER);
   processHits (hits, selectBuf);
}
```

```
void drawWorldScene (bool selection_mode)
{
   // initialize names stack
   if (selection_mode)
   {
      glInitNames ();
      glPushName (-1);
   }

   // green triangle has name 1
   glColor3f (0.0, 1.0, 0.0);
   if (selection_mode) glLoadName (1);
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -5.0);

   // red triangle has name 2
   glColor3f (1.0, 0.0, 0.0);
   if (selection_mode) glLoadName (2);
   drawTriangle (2.0, 7.0, 3.0, 7.0, 2.5, 8.0, -5.0);

   // yellow triangles have name 3
   glColor3f (1.0, 1.0, 0.0);
   if (selection_mode) glLoadName (3);
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, 0.0);
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -10.0);
}
```

Selection example

```
void processHits (GLint hits, GLuint buffer[])
{
   unsigned int i, j;
   GLuint names, *ptr;

   printf (“Number of hits = %d\n", hits);
   ptr = (GLuint *) buffer;
   for (i = 0; i < hits; i++)
   {
      names = *ptr;
      printf (“Values for hit no. %d\n", i);
      printf ("  number of names in stack = %d\n", names); ptr++;
      printf ("  z1 = %u\n", *ptr); ptr++;
      printf (“  z2 = %u\n", *ptr); ptr++;
      printf ("  the name stack: ");
      for (j = 0; j < names; j++)
      {
         printf ("%d, ", *ptr); ptr++;
      }
      printf ("\n");
   }
}
```

Picking

- Determining objects that are under cursor
- Triggered usually by mouse button
- Small rectangle (picking area) is created around cursor position
- Picking area is extended into 3D creating truncated pyramid
- Determining all objects that are inside truncated pyramid – using selection mode

Picking matrix

- **void gluPickMatrix(GLdouble x, GLdouble y, GLdouble width, GLdouble height, GLint viewport[4])**
  - Creates addition to projection matrix that restricts drawing to a small region of the viewport and multiplies that matrix with current matrix
  - (x,y) – center of picking region
  - width, height – size of picking area
  - viewport – viewport used to determine relative position of picking area

Picking example

```
void mouse_click_callback(int button, int state, int x, int y)
{
   if (button == 0 && state == 0)
   {
      GLint width = 5;
      GLint height = 5;
      GLint viewport[4];
      glGetIntegerv (GL_VIEWPORT, viewport);

      // prepare projection matrix with picking and view matrix
      glMatrixMode (GL_PROJECTION);
      glLoadIdentity ();
      gluPickMatrix (x, viewport[3] - y, width, height, viewport);
      gluPerspective ();
      glMatrixMode (GL_MODELVIEW);
      glLoadIdentity ();
      gluLookAt (7.5, 7.5, 12.5, 0, 0, 0, 0.0, 1.0, 0.0);

      // reder scene in selection mode using prepared transformations
      GLuint selectBuf[BUFSIZE];
      glSelectBuffer (BUFSIZE, selectBuf);
      glRenderMode (GL_SELECT);
      drawWorldScene (true);
      glFlush ();

      // return to normal mode and write all objects under cursor
      GLint hits = glRenderMode (GL_RENDER);
      processHits (hits, selectBuf);
   }
}
```

The End!

Questions?