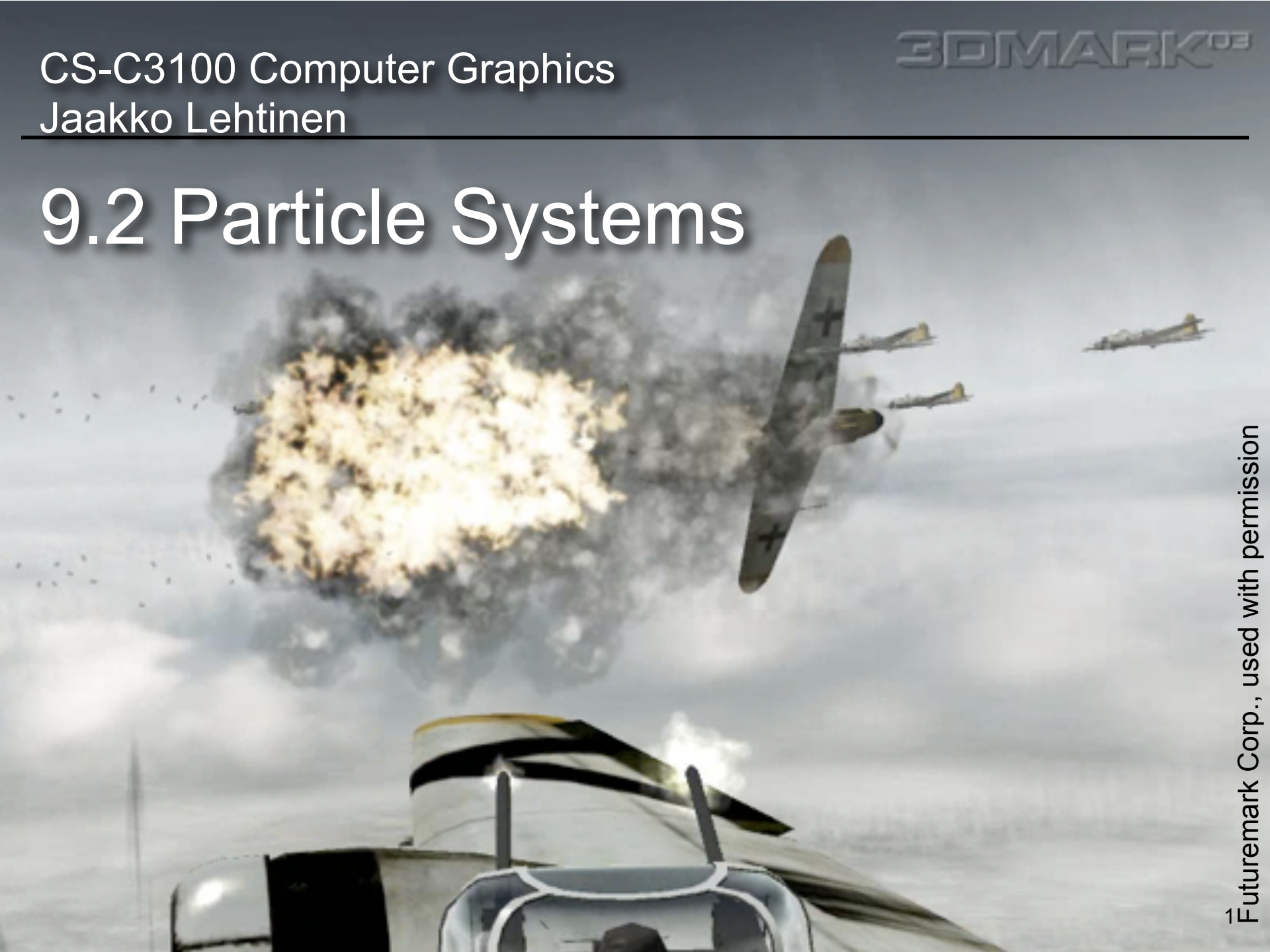
9.2 Particle Systems

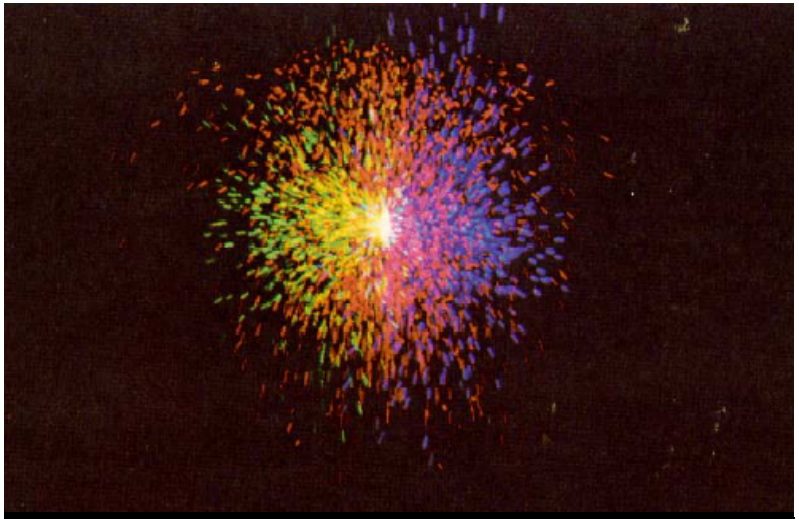


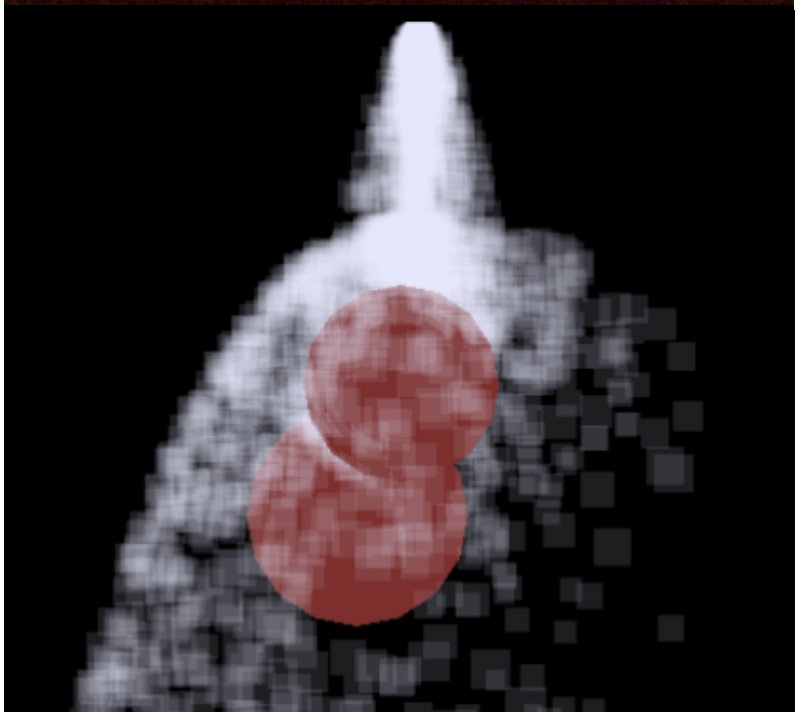
In This Video

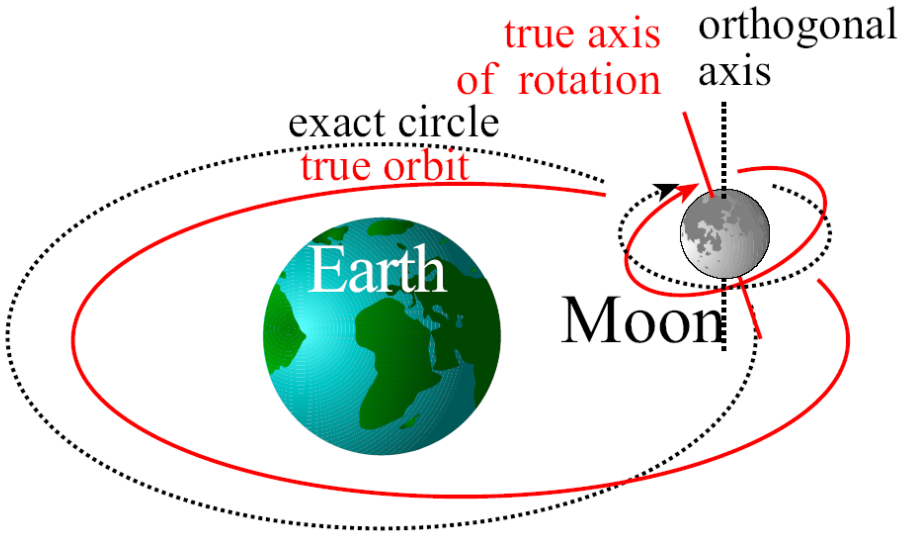
- Particle systems: the simplest physically-inspired animation techniques

Particles: Focus on Point Dynamics

- Lots of points!
- "Particle systems"
  - Borderline between procedural and physically-based







Link

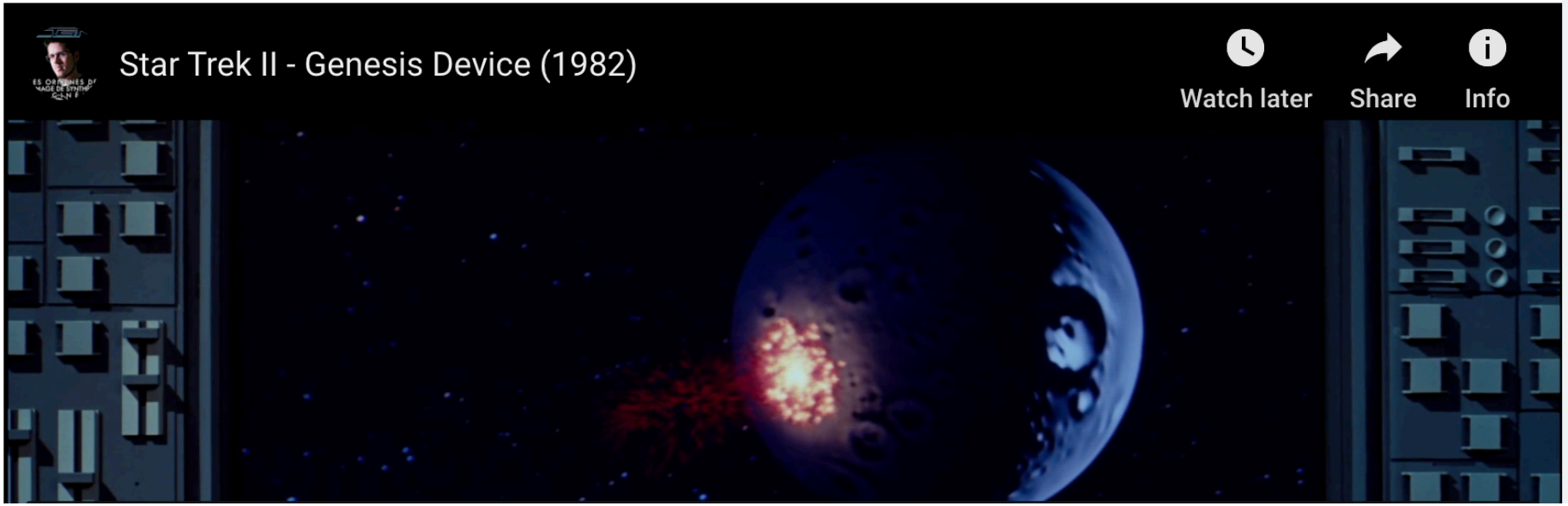
Posted by u/CaptainBroverdose 3 years ago

245

The "Genesis sequence" from 'Star Trek II: The Wrath Of Khan' (1982) was the first completely computer-generated sequence in a film.

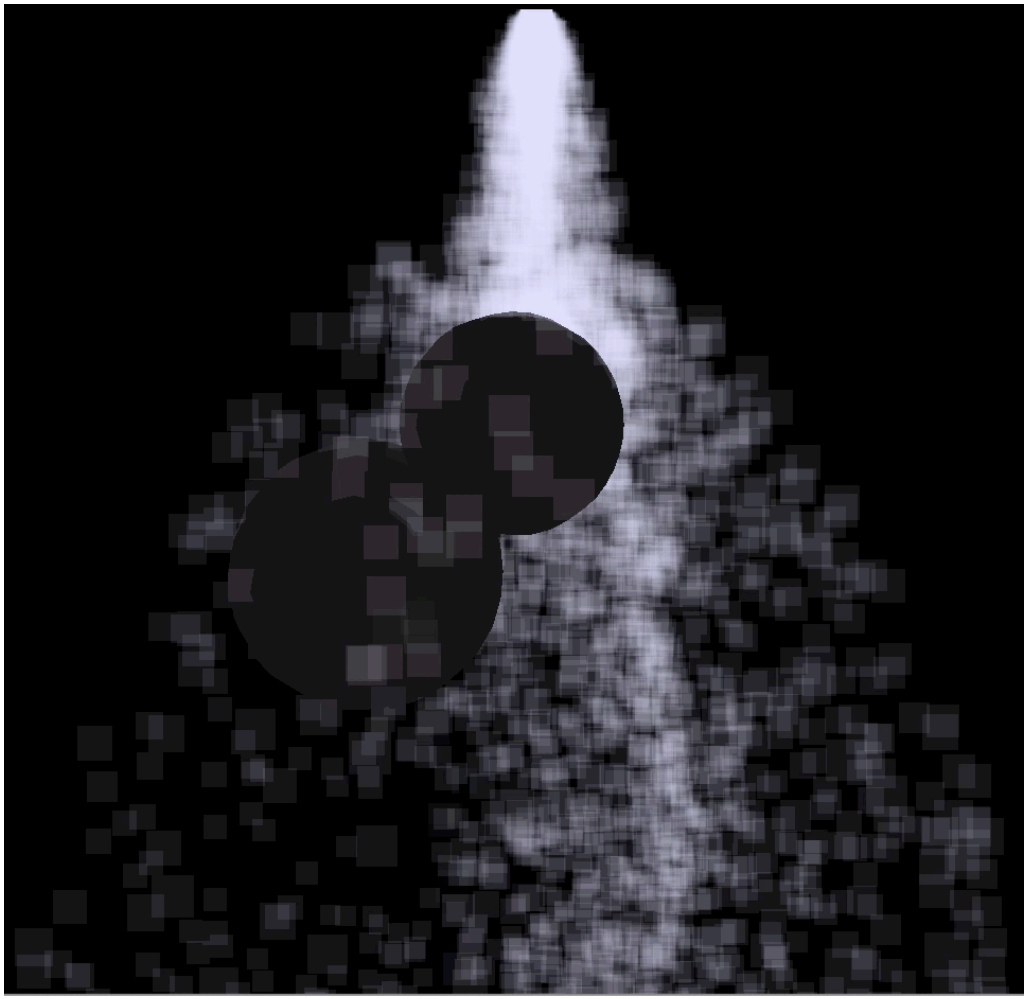
youtube.com/watch?...

Media



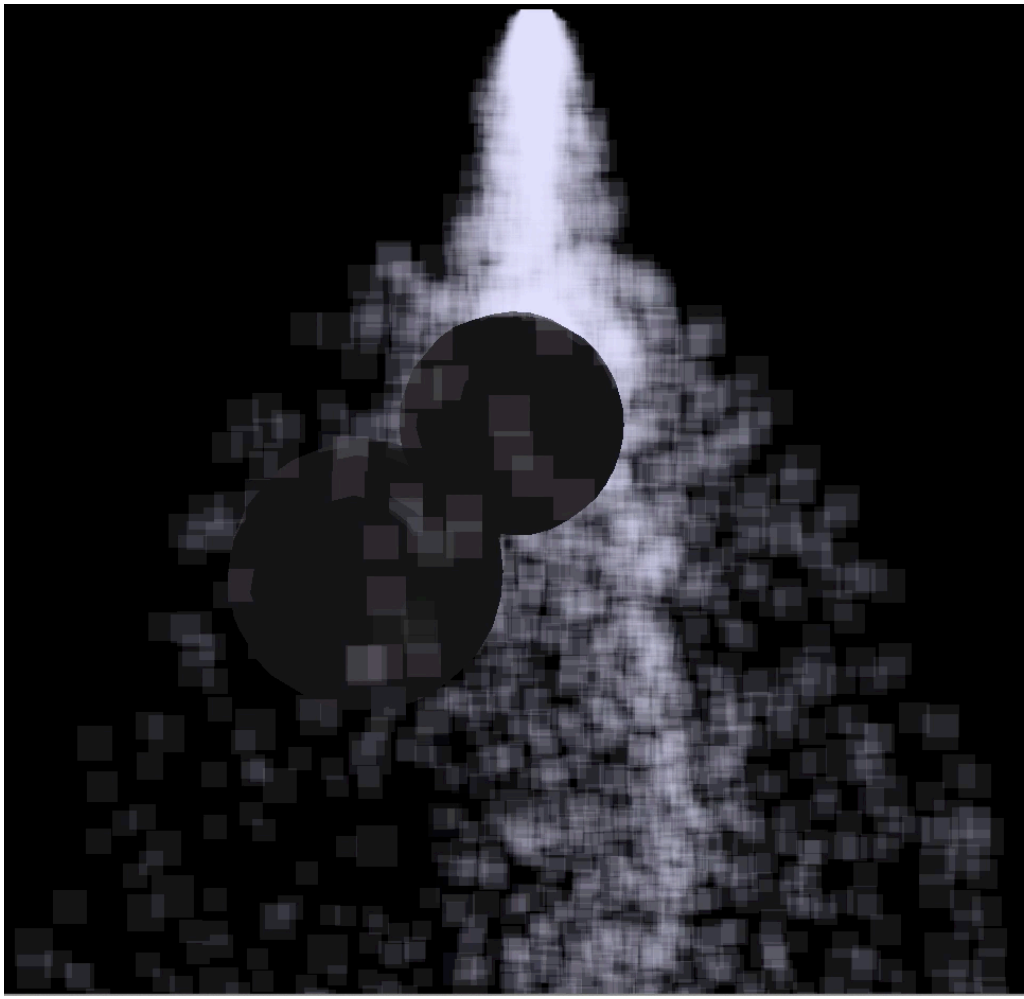
Demo

- Simple “waterfall”
  - Each particle affected by gravity
  - Simple obstacle geometry (spheres)
  - Braindead rendering (screen-aligned transparent quads)



Demo

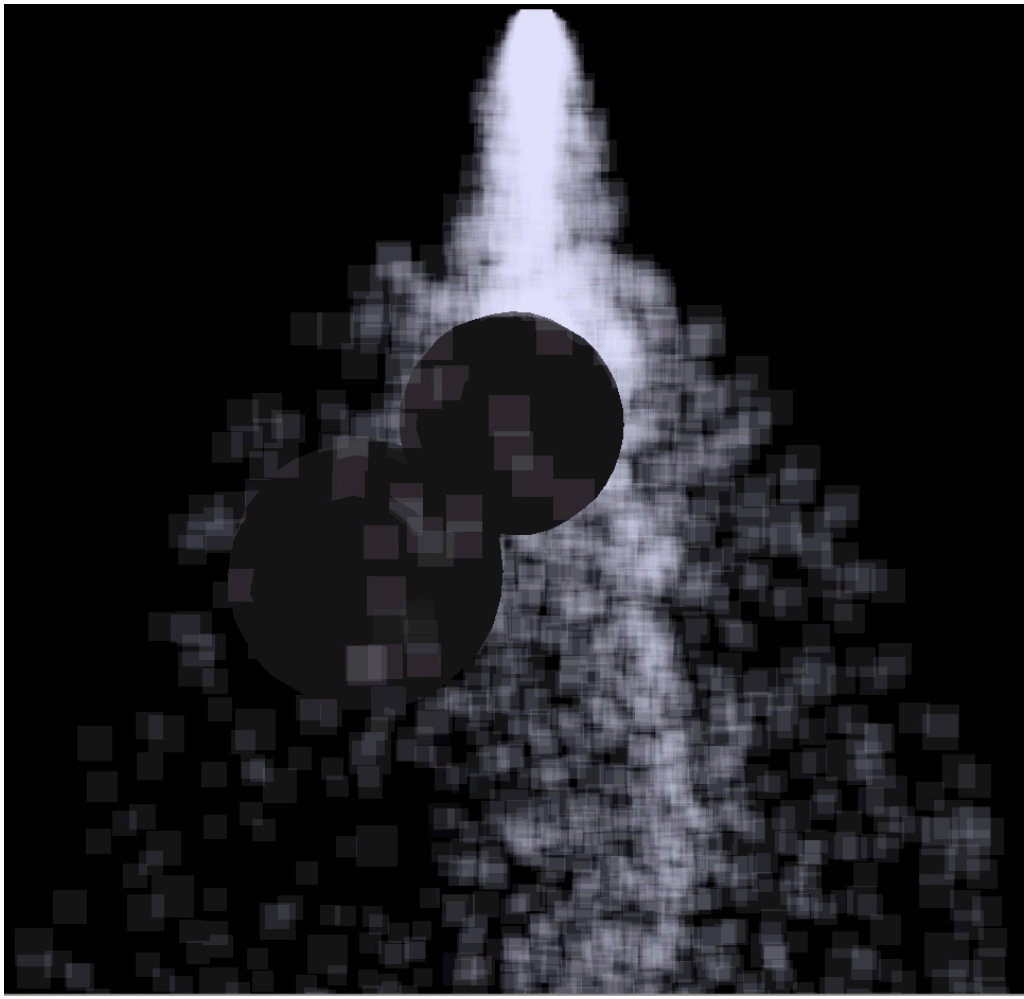
- Simple “waterfall”
  - Each particle affected by gravity
  - Simple obstacle geometry (spheres)
  - Braindead rendering (screen-aligned transparent quads)



Demo

- Simple “waterfall”
  - Each particle affected by gravity
  - Simple obstacle geometry (spheres)
  - Braindead rendering (screen-aligned transparent quads)

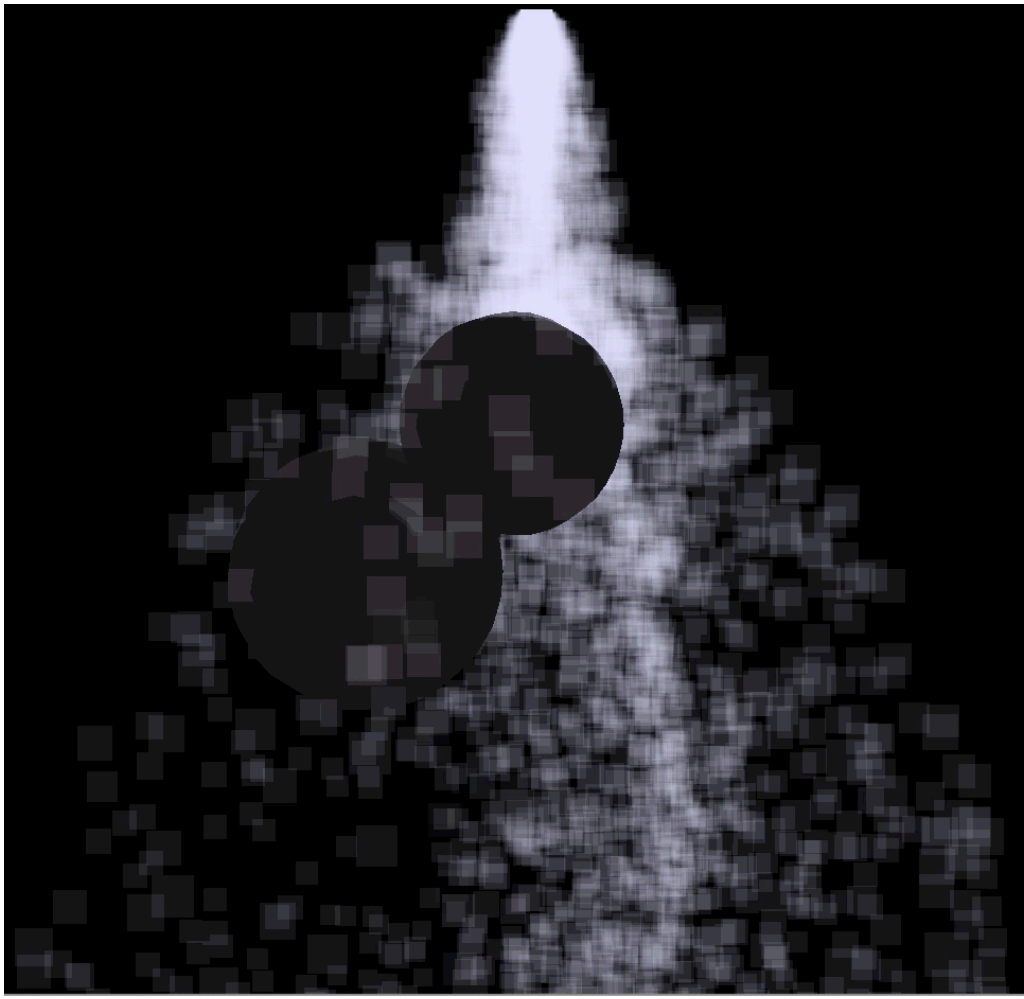
Note: No interaction between particles. It’s not a fluid simulation! But then again, it took under an hour to code up.



Demo

- Simple “waterfall”
  - Each particle affected by gravity
  - Simple obstacle geometry (spheres)
  - Braindead rendering (screen-aligned transparent quads)

Note: No interaction between particles. It’s not a fluid simulation! But then again, it took under an hour to code up.

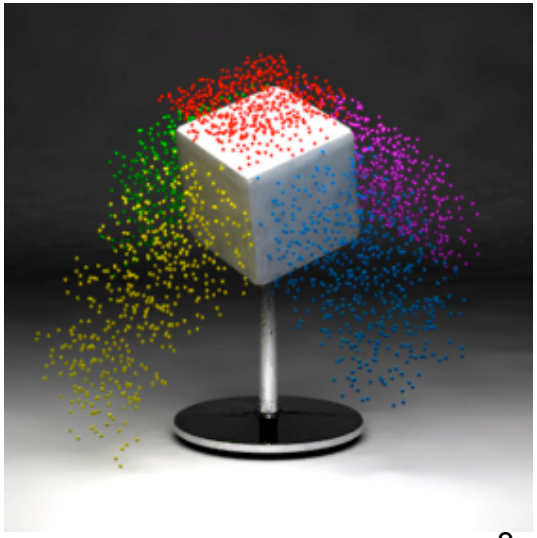


Remember Demo from Last Time

Futuremark Corp., used with permission

What is a Particle System?

- Collection of many small simple pointlike things
  - Described by their current state: position, velocity, age, color, etc.
- Particle motion influenced by external force fields and internal forces between particles
- Particles created by **generators** or **emitters**
  - With some randomness
- Particles often have lifetimes
- Particles are often independent
- Treat as points for dynamics, but rendered as anything you want



http://en.wikipedia.org/wiki/Image:Particle_Emitter.jpg

Simple particle system: sprinkler

```
PL: linked list of particle = empty;
```

Image Jeff Lander

Simple particle system: sprinkler

```
PL: linked list of particle = empty;
spread=0.1;//how random the initial velocity is
colorSpread=0.1; //how random the colors are
```

Image Jeff Lander

Simple particle system: sprinkler

```
PL: linked list of particle = empty;
spread=0.1;//how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
```

Image Jeff Lander

Simple particle system: sprinkler

```
PL: linked list of particle = empty;
spread=0.1;//how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
    Kill particles that are too old
    Generate new particles: loop
        p=new particle();
        p->position=(0,0,0);
        p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
        p.color=(0,0,1)+colorSpread*(rnd(), rnd(),rnd());
        PL->add(p);
```

Image Jeff Lander

Simple particle system: sprinkler

```
PL: linked list of particle = empty;
spread=0.1;//how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
    Kill particles that are too old
    Generate new particles: loop
        p=new particle();
        p->position=(0,0,0);
        p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
        p.color=(0,0,1)+colorSpread*(rnd(), rnd(),rnd());
        PL->add(p);
    For each particle p in PL
        p->position+=p->velocity*dt; //dt: time step
        p->velocity+=g*(0,0,-1)*dt; //g: gravitation constant
        glColor(p.color);
        glVertex(p.position);
```

Image Jeff Lander

Simple particle system: sprinkler

```
PL: linked list of particle = empty;
spread=0.1;//how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
    Kill particles that are too old
    Generate new particles: loop
        p=new particle();
        p->position=(0,0,0);
        p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
        p.color=(0,0,1)+colorSpread*(rnd(), rnd(),rnd());
        PL->add(p);
    For each particle p in PL
        p->position+=p->velocity*dt; //dt: time step
        p->velocity+=g*(0,0,-1)*dt; //g: gravitation constant
        glColor(p.color);
        glVertex(p.position);
```

Image Jeff Lander

Simple particle system: sprinkler

```
PL: linked list of particle = empty;
spread=0.1;//how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
    Kill particles that are too old
    Generate new particles: loop
        p=new particle();
        p->position=(0,0,0);
        p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
        p.color=(0,0,1)+colorSpread*(rnd(), rnd(),rnd());
        PL->add(p);
    For each particle p in PL
        p->position+=p->velocity*dt; //dt: time step
        p->velocity+=g*(0,0,-1)*dt; //g: gravitation constant
        glColor(p.color);
        glVertex(p.position);
```

Image Jeff Lander

Corresponding Waterfall Demo Code

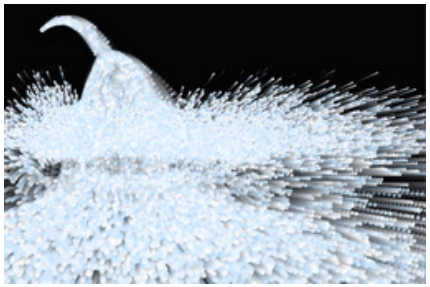
```
void update( float dt )
{
    // move sphere...
    static float t = 0.0f;
    t += dt;
    float x = sin( t );
    ((SphereBlocker*)*(m_blockers.begin()))->center().x = -0.5f + x;

    for ( list<Particle*>::iterator i = m_alive.begin(); i != m_alive.end(); ++i )
    {
        Particle* pP = *i;

        float fGravity = -dt * 9.81;
        pP->v.y += fGravity;

        pP->p += dt * pP->v;

        pP->lifetime += dt;

        for ( list<Blocker*>::iterator b = m_blockers.begin(); b != m_blockers.end(); ++b )
            (*b)->testAndHandleCollision( pP );

        if ( pP->lifetime > 3.0f )
        {
            m_storage.push_back( pP );
            i = m_alive.erase( i );
        }
    }

    emit();
}
```

Where do particles come from?

- Created by generators or **emitters**
  - Can be attached to objects in the model
- Given rate of creation: particles/second
  - record t_{last} of last particle created

$$n = \lfloor (t - t_{last})\, rate \rfloor$$

  - create n particles.
    update t_{last} if $n > 0$



http://www.particlesystems.org/

- Create with (random) distribution of initial $\boldsymbol{x}$ and $\boldsymbol{v}$
  - if creating $n > 1$ particles at once, spread out on path

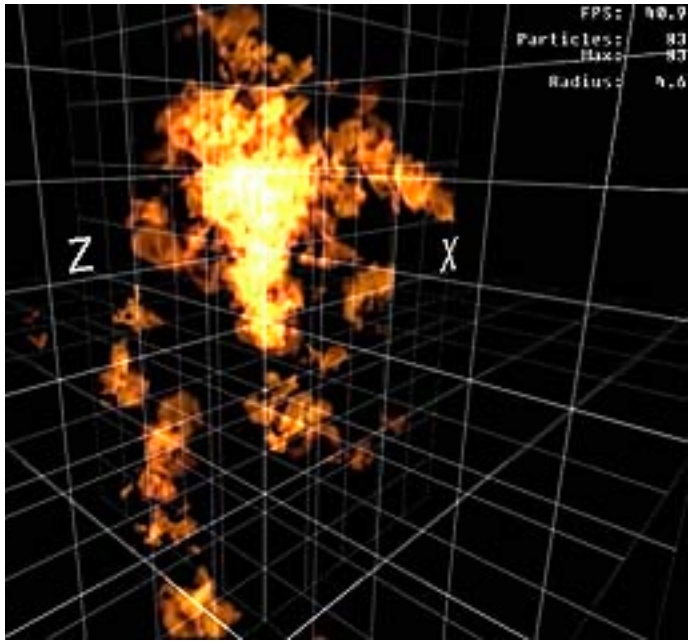
Code from Waterfall Demo

```
void emit()
{
    const int iMaxEmissionsPerFrame = 20;
    int iEmissions = 0;

    Vector3 vEmitterPos( 0.0f, 3.0f, -0.0f );

    while ( !m_storage.empty() && iEmissions < iMaxEmissionsPerFrame )
    {
        Particle* pNew = m_storage.back();
        m_storage.pop_back();

        pNew->p = vEmitterPos;
        float angle = rand_01() * 2.0f * PI;
        float angle2 = rand_01();
        pNew->v.x = angle2 * sin( angle );
        pNew->v.z = angle2 * cos( angle );
        pNew->v.y = -(1.0f - angle2);
        pNew->v *= 1.05f;

        pNew->lifetime = 0.0f;

        m_alive.push_back( pNew );

        ++iEmissions;
    }
}
```

Particle Controls

MAX PAYNE 2

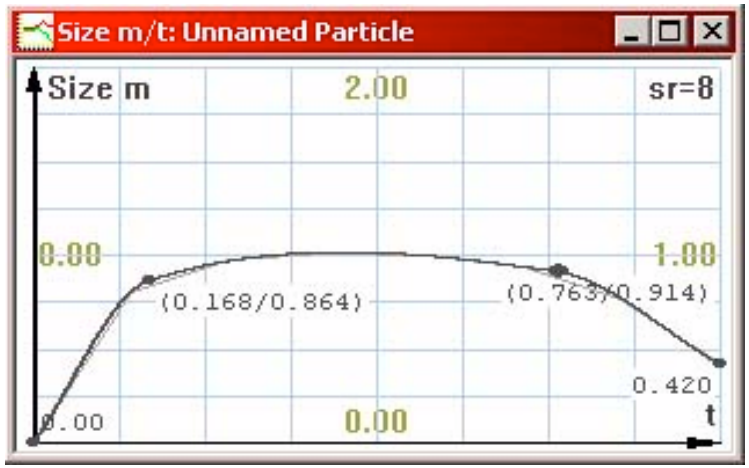
- In production tools, all these variables are time-varying and controllable by the user (artist)
  - Emission rate, color, velocity distribution, direction spread, textures, etc. etc.
    - All as a function of time!
  - Example: ParticleFX (Max Payne Particle Editor)
    - Custom editor software
    - You can **download it** (for Windows) and easily create your own particle systems. Comes with examples!
    - This is what we used for all the particles in the game

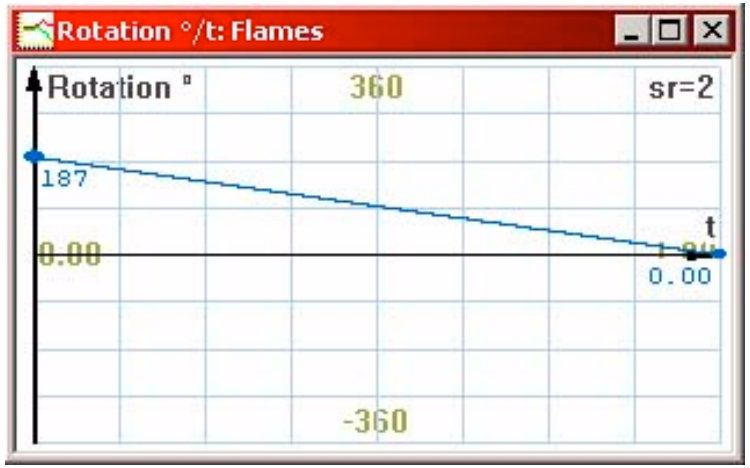


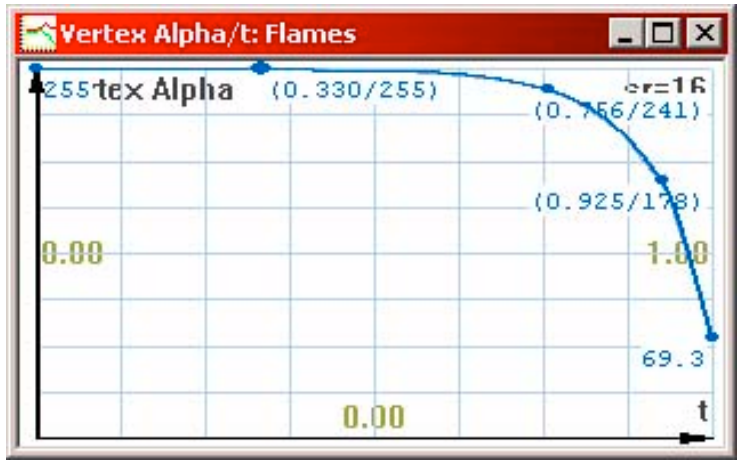
Emitter Controls

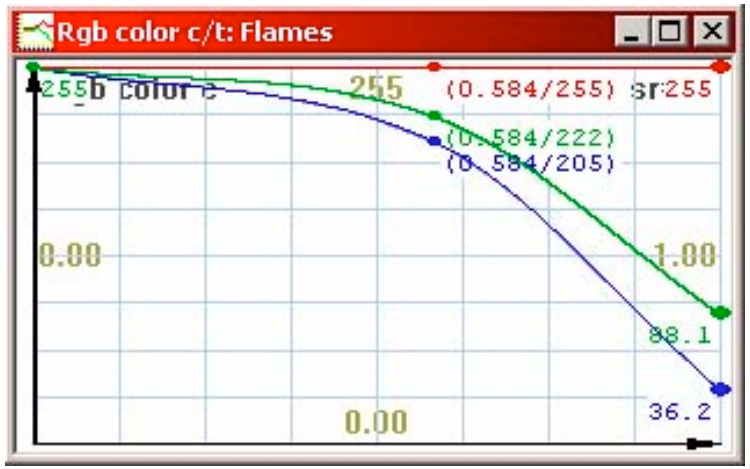
MAX PAYNE 2

- Again, reuse splines!







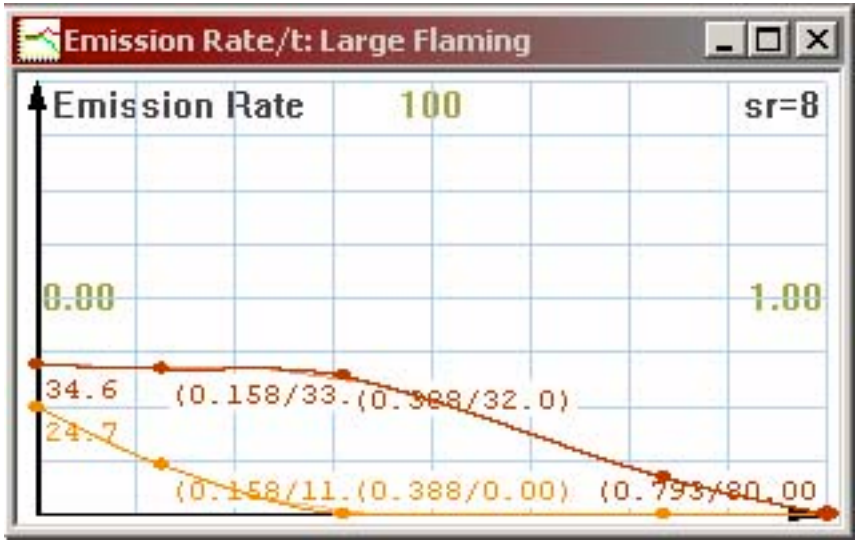


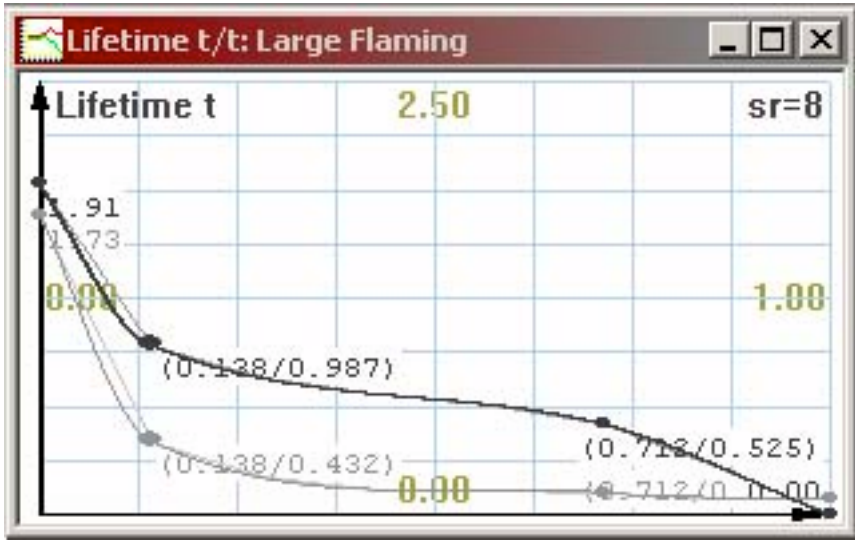
Controls from ParticleFX by Remedy Entertainment

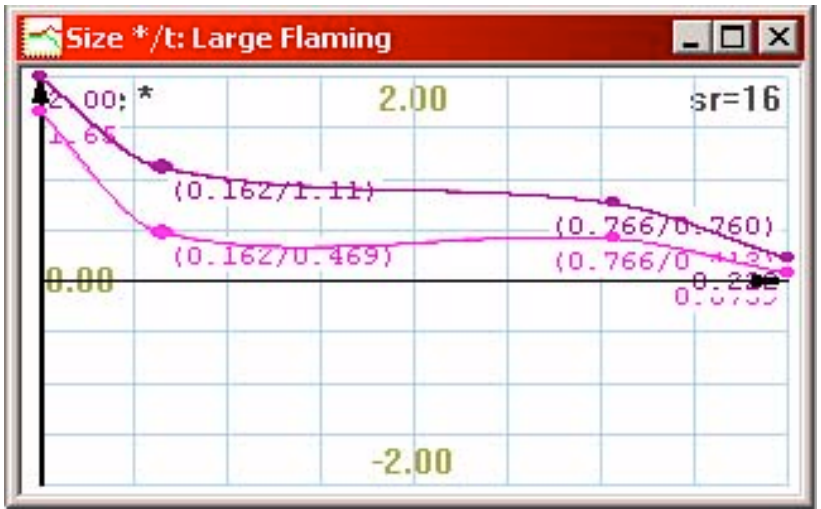
Emitter Controls

MAX PAYNE 2

- Again, reuse splines!







Controls from ParticleFX by Remedy Entertainment

Particle Lifetimes

- Initialize particle age to 0 at emission
- Keep track of its age over simulation
- Use particle age to
  - Remove particles from system when too old
  - Change color or animate texture
    (flame, puff of smoke)
  - Change transparency (old particles fade)

- Perhaps also remove particles that are offscreen
  - Careful with this, it only works if the camera is static!

Rendering and Motion Blur

- Often not shaded (just emission, think sparks)
  - But realistic non-emissive particles needs shadows, etc.
- Most often, particles don't contribute to the z-buffer, i.e., they do not fully occlude stuff that's behind
  - Rendered with z testing on
    (particles get occluded by solid stuff)
- Draw a line for motion blur
  - (**x**, **x**+**v** d*t*)
  - Or an elongated quad with texture

Metal Gear Solid by Konami

Rendering and Motion Blur

Metal Gear Solid by Konami

Rendering and Motion Blur

- Often use texture maps (fire, clouds, smoke puffs)
  - Called “billboards” or “sprites”
  - Always parallel to image plane

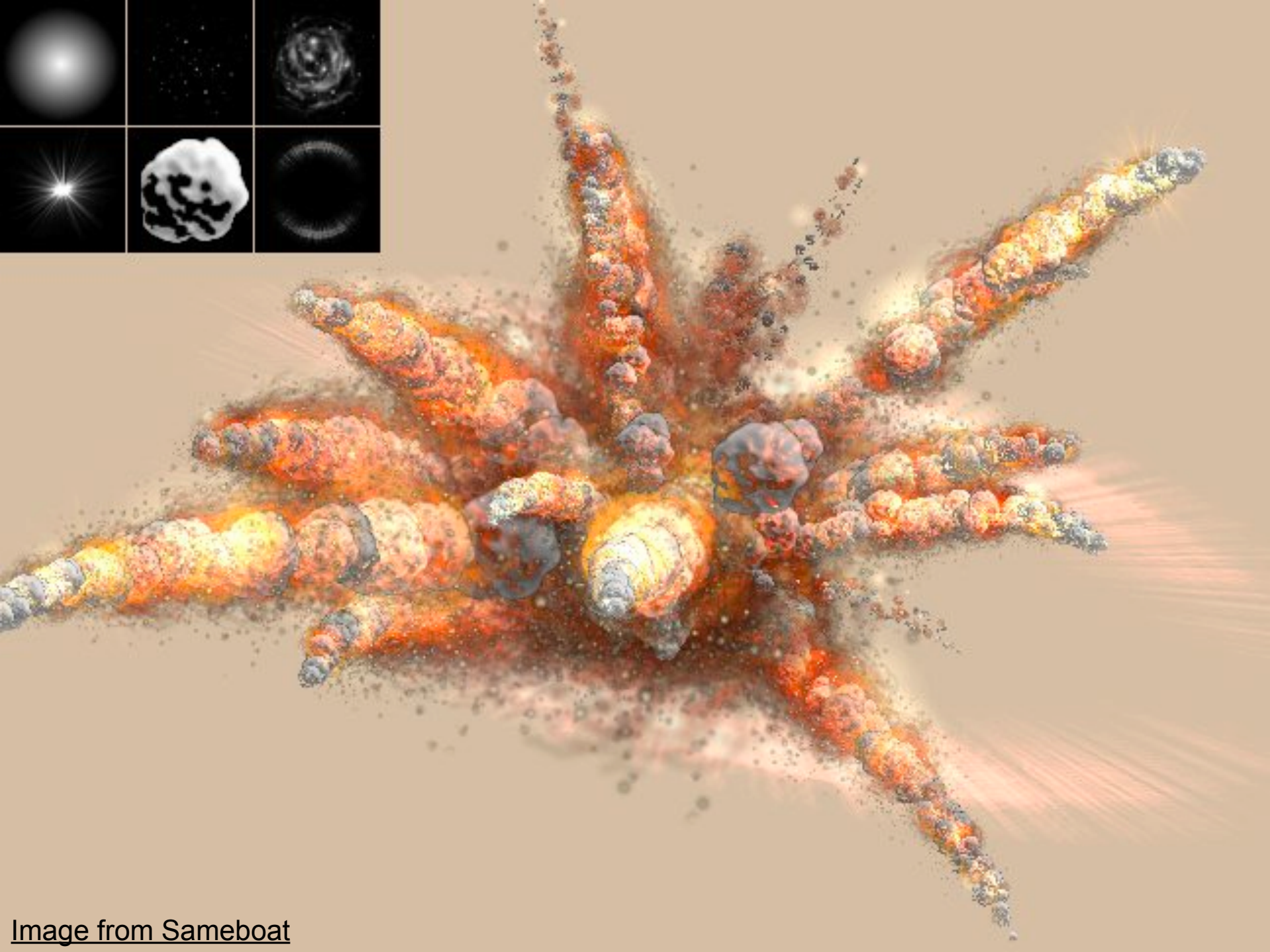
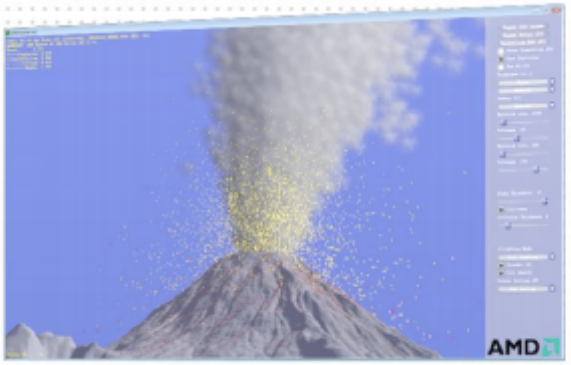


Image from Sameboat

Particle Systems Entirely on GPU

GDC 14

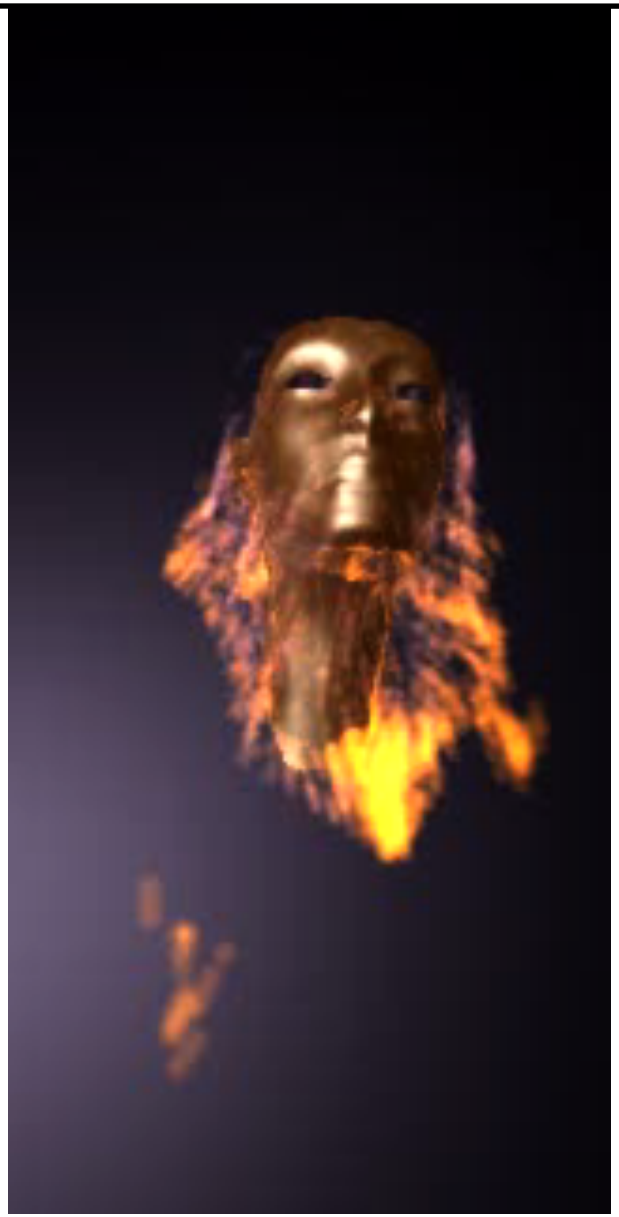
Compute-Based GPU Particle Systems



Gareth Thomas
Developer Technology Engineer, AMD

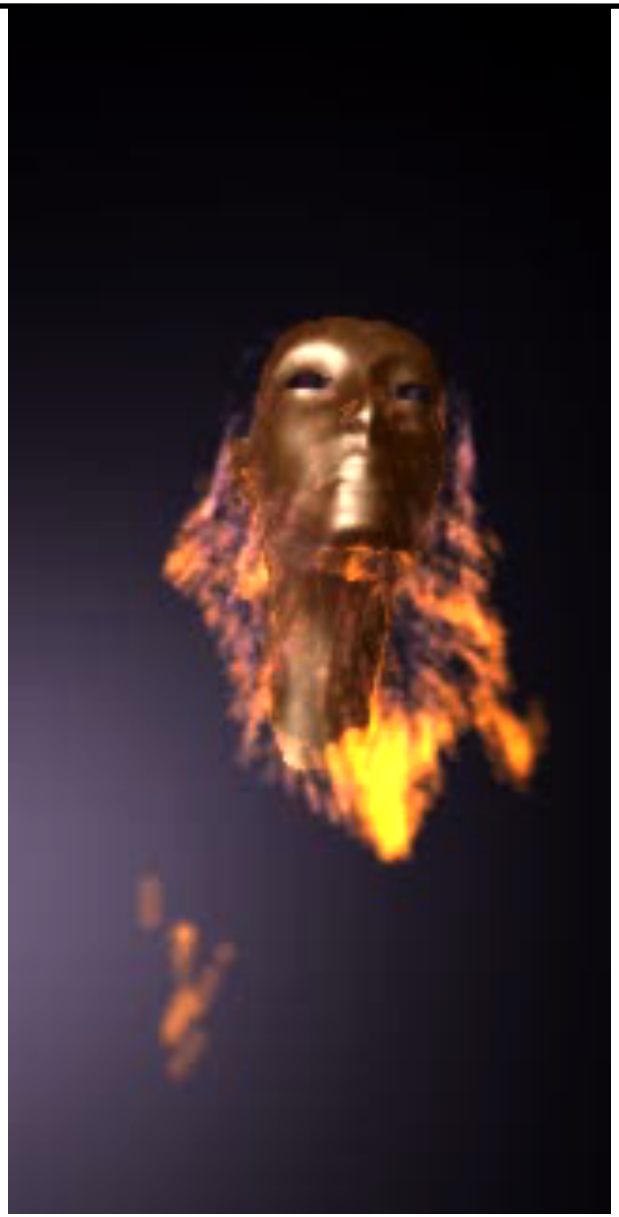
GAME DEVELOPERS CONFERENCE 2014
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21

That’s all..



- **Next, dynamics: how the system evolves**

That’s all..



- **Next, dynamics: how the system evolves**