PHYSICS

Ross Hays, Isaac Folzenlogen,
John Chumley, Zachary Atwood

Physics Engines

Why are physics engine needed

- Many games what some response in game to collisions and other interactions
- Programming every possible interaction in infeasible
- Reusable between multiple games

Limitations

- Real time physics engines such as those used in games cannot simulate perfect interactions
- Some shortcuts are needed to keep processing time down for real time
- Too many objects to simulate also results in slowdown

Some popular physics engines

NVDIA PhysX (used in Unity & Unreal)
Havok (Half-Life 2, Halo series, Skyrim)
Box2D (2D physics engine, Angry Birds)
Bullet (open source, shown later)

Bouncing ball demonstration

Bouncing

```
using UnityEngine;
using System.Collections;

public class BouncingBall : MonoBehaviour {
    // populate the scene with all needed objects
    void Start () {
        // create and position the ground plane
        GameObject ground = GameObject.CreatePrimitive(PrimitiveType.Plane);
        ground.name = "Ground";
        ground.transform.localScale = new Vector3(100, 1, 100);
        ground.transform.position = new Vector3(0, 0, 0);
        // create and position the bouncing ball
        GameObject ball = GameObject.CreatePrimitive(PrimitiveType.Sphere);
        ball.name = "Bouncing ball";
        ball.transform.position = new Vector3(0, 3, 0);
        // give the ball a bouncy material for physics
        PhysicMaterial bouncyMaterial = new PhysicMaterial();
        bouncyMaterial.bounciness = 1;
        bouncyMaterial.bounceCombine = PhysicMaterialCombine.Maximum;
        ball.GetComponent<Collider>().material = bouncyMaterial;
        // add the game objects to the PhysX world by attaching rigidbodies
        Rigidbody groundBody = ground.AddComponent<Rigidbody>();
        Rigidbody ballBody = ball.AddComponent<Rigidbody>();
        // ground doesn't need updated the same way the ball does, mark kinematic
        groundBody.isKinematic = true;
    }
}
```

Implementation: Phases & Solvers

- Two main components of most physics engines: collision detection and collision resolution
- Collision detection broken into multiple phases
- Collision resolution uses multiple solvers

Collision Detection

Broad Phase: Basically what it sounds like, preliminary BB check or sweep and prune

Narrow Phase: Exact collisions checked for if the possibility made it past the broad phase

Collision Resolution

- Solvers are algorithms that exist to decide the proper response to all collisions that made it past detection
- Iterative solvers iteratively improve
- Allow callbacks on specific items for user handled collisions

Collision demonstration

Collision demonstration

```
6     void CreateBoxPyramid(int dimension) {
7         GameObject pyramid = new GameObject("pyramid");
8         for (int y = 0; y < dimension; y++) {
9             // center for this layer of the pyramid
10            float layerHeight = 0.5f + y;
11            int layerCount = dimension - y;
12            for (int x = 0; x < layerCount; x++) {
13                for (int z = 0; z < layerCount; z++) {
14                    // figure out where we are putting the next box
15                    Vector3 boxCenter = new Vector3();
16                    boxCenter.x = x - layerCount / 2.0f;
17                    boxCenter.y = layerHeight;
18                    boxCenter.z = z - layerCount / 2.0f;
19                    // create and position the box (as a rigidbody)
20                    GameObject box = GameObject.CreatePrimitive(PrimitiveType.Cube);
21                    box.name = "Box[" + x + "," + y + "," + z + "]";
22                    box.transform.parent = pyramid.transform;
23                    box.transform.position = boxCenter;
24                    box.AddComponent<Rigidbody>();
25                }
26            }
27        }
28    }
```

```
30      void Start () {
31          // create and position the ground plane, nothing new here
32          GameObject ground = GameObject.CreatePrimitive(PrimitiveType.Plane);
33          ground.name = "Ground";
34          ground.transform.localScale = new Vector3(100, 1, 100);
35          ground.transform.position = new Vector3(0, 0, 0);
36          Rigidbody groundBody = ground.AddComponent<Rigidbody>();
37          groundBody.isKinematic = true;
38          // place boxes in a pyramid shape
39          CreateBoxPyramid(12);
40          // ITERATIVE SOLVER ITERATIONS
41          Physics.solverIterationCount = 7; // 7 is the Unity default
42      }
43
44      // launch a sphere at the point clicked on
45      void Update() {
46          if (Input.GetMouseButtonDown(0)) {
47              Camera cam = GameObject.Find("Main Camera").GetComponent<Camera>();
48              Ray ray = cam.ScreenPointToRay(Input.mousePosition);
49              Vector3 dest = ray.origin + ray.direction * 20;
50              GameObject projectile = GameObject.CreatePrimitive(PrimitiveType.Sphere);
51              projectile.transform.position = cam.transform.position;
52              projectile.transform.localScale = new Vector3(0.5f, 0.5f, 0.5f);
53              Rigidbody projBody =  projectile.AddComponent<Rigidbody>();
54              projBody.useGravity = false;
55              projBody.mass = 1000;
56              projBody.AddForce(projBody.mass * 100 * (dest - projectile.transform.position));
57          }
58      }
```

```
30      void Start () {
31          // create and position the ground plane, nothing new here
32          GameObject ground = GameObject.CreatePrimitive(PrimitiveType.Plane);
33          ground.name = "Ground";
34          ground.transform.localScale = new Vector3(100, 1, 100);
35          ground.transform.position = new Vector3(0, 0, 0);
36          Rigidbody groundBody = ground.AddComponent<Rigidbody>();
37          groundBody.isKinematic = true;
38          // place boxes in a pyramid shape
39          CreateBoxPyramid(12);
40          // ITERATIVE SOLVER ITERATIONS
41          Physics.solverIterationCount = 7; // 7 is the Unity default
42      }
43
44      // launch a sphere at the point clicked on
45      void Update() {
46          if (Input.GetMouseButtonDown(0)) {
47              Camera cam = GameObject.Find("Main Camera").GetComponent<Camera>();
48              Ray ray = cam.ScreenPointToRay(Input.mousePosition);
49              Vector3 dest = ray.origin + ray.direction * 20;
50              GameObject projectile = GameObject.CreatePrimitive(PrimitiveType.Sphere);
51              projectile.transform.position = cam.transform.position;
52              projectile.transform.localScale = new Vector3(0.5f, 0.5f, 0.5f);
53              Rigidbody projBody =  projectile.AddComponent<Rigidbody>();
54              projBody.useGravity = false;
55              projBody.mass = 1000;
56              projBody.AddForce(projBody.mass * 100 * (dest - projectile.transform.position));
57          }
58      }
```

Collision Detection methods

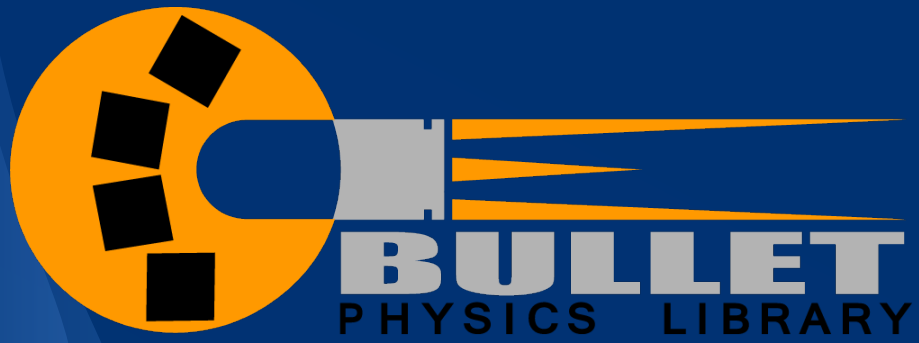
- There are two main detection methods of broadphase, discrete and continuous
- Discrete checks on each game frame
- Continuous checks path the object took in between frames, more expensive

Continuous vs Discrete demo

Forces and Constants

- In addition to collision handling in most engines
- Useful for things like air drag, gravity, magnetism (recalculated every time)

Bullet Physics Engine



About Bullet Physics

- Open source (zlib license)
- First created in 2003
- Written in C/C++
- Ported to Java, C#, Javascript, and more
- Used in Blender game engine

An introduction

- Many parallels with Unity's physics system (PhysX behind the scenes)
- Rigid bodies are items in physics simulation
- Rigid bodies exist in a defined world object
- Divided in Bullet Collision and Dynamics

PhysX

About PhysX

- Multi-threaded physics simulation SDK
- Developed in 2004 by Ageia - used PPU
- Acquired by Nvidia in 2008 - all builds after 2.8.3 use GPU

Can you use it?

- Proprietary (non-free), except:
  - Free for Windows developers
  - Free for educational and non-commercial use on Linux, OS X, and Android

Havok

About Havok

- By company of the same name
- First released at GDC 2000
- Uses dynamic constraints on rigid bodies for ragdoll physics
- 2008 Released version 6.5

Other Havok Releases

- 2008 Havok Cloth
- 2008 Havok Destruction

Which Engine is Right for Me?

- Physics Engine Evaluation Lab (PEEL) compared engines memory consumption
  - Bullet 2.8.1 is worse than PhysX 3.3
  - PhysX 2.8.4 and earlier was substantially better at sweep tests than Bullet
- PhysX consumes more GPU, Havok consumes more CPU

Brownian Motion

What is Brownian Motion?

- Equations used to study turbulent motion in colloids and macromolecular fluids

Diffusive Brownian Motion

- Used when large particles diffuse slowly through fluids
- Assume inertia = 0

$$x_{n+1} = x_n + v^s \delta t + A r_n$$

$$A = (24 D \delta t)^{1/2}$$

Lagrangian formulation of Newtons Equations

- Used when particles move quickly
- considers inertia
- Use fluctuation dissipation theorem to calculate random force

$$\langle f^r(t_1) f^r(t_2) \rangle = 2kT\zeta\delta(t_1 - t_2)$$

$$\delta(t_1, t_2) = \begin{cases} 1/\delta t & \text{if } t_1 \text{ and } t_2 \text{ are in the same time-step of } \delta t \\ 0 & \text{otherwise} \end{cases}$$

Lagrangian (continued)

Apply random force to Newton's formula

$$m\ddot{x} + \zeta\dot{x} = f^s + f^r$$

Solve for position

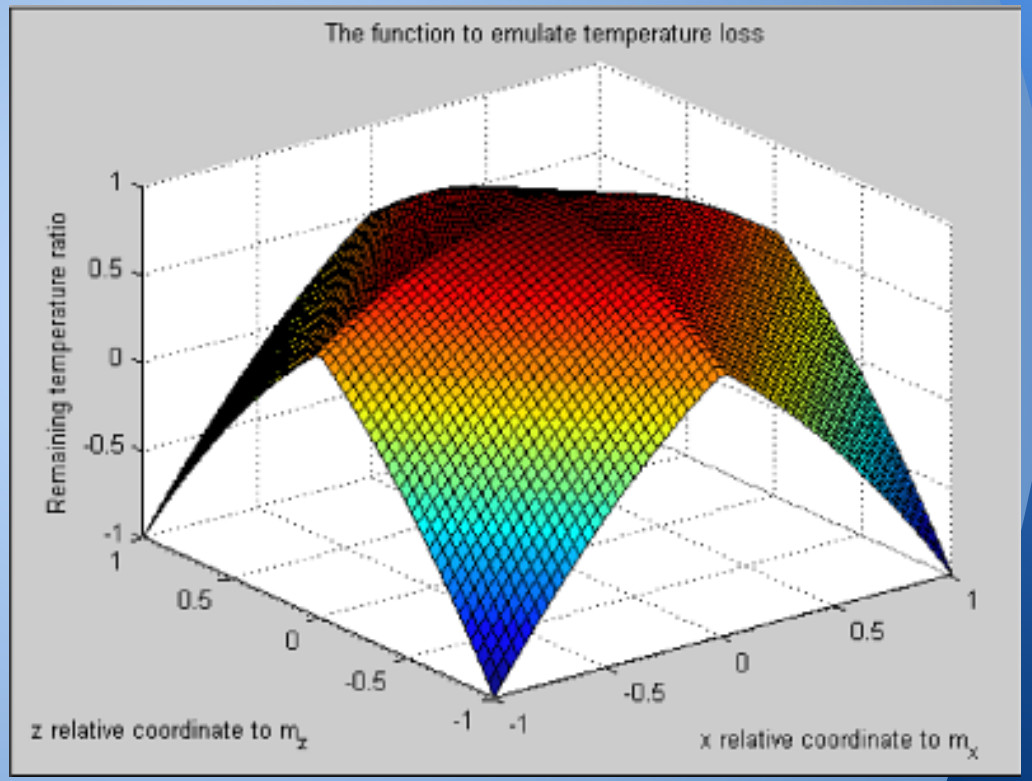
$$\begin{aligned} x_{n+1} &= x_n + \dot{x}_n \delta t, \\ \dot{x}_{n+1} &= \dot{x}_n + m^{-1}\left(-\zeta\dot{x}_n + f^s + (24kT\zeta/\delta t)^{1/2} r_n\right)\delta t \end{aligned}$$

Fire Particle Effect

- Select “source” of particles using Gaussian
- Three forces:
  - Thermal Buoyancy (proportionate to temp)
  - Wind
  - Brownian Forces (turbulence)

Temperature Decay

- T initially 1500 ℃
- T<500℃ reinitialize
- T decays as distance from origin increases



Smoothed Particle Hydrodynamics

- Used to “smooth” area between particles for fluids (e.g. smoke and water)
- Kernal function W (e.g. Gaussian, Cubic)
- Calculates A(r) for any point r (not necessarily a particle)

A(r)

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\mathbf{r} - \mathbf{r}_j|, h)$$

$$\rho_i = \rho(\mathbf{r}_i) = \sum_j m_j \frac{\rho_j}{\rho_j} W(|\mathbf{r}_i - \mathbf{r}_j|, h) = \sum_j m_j W(\mathbf{r}_i - \mathbf{r}_j, h)$$

Game Engine Simulation of Soft Bodies

Soft Body Dynamics

- Computer Graphics (plausible not necessarily physically accurate)
- Different from
  - Rigid bodies (no relative internal movement)
  - Fluids (constantly deformable)
- Examples Include
  - Muscles
  - Hair
  - Vegetation
  - Cloth

Modeling Techniques

- Spring Masses
- Energy Minimization

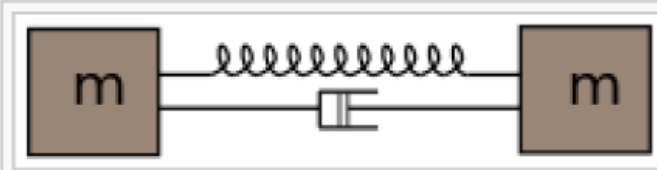
  Low Resolution

  Computationally Very Expensive

- Finite element simulation
  - Tetrahedral mesh
  - Computationally Expensive
- Rigid Body Based

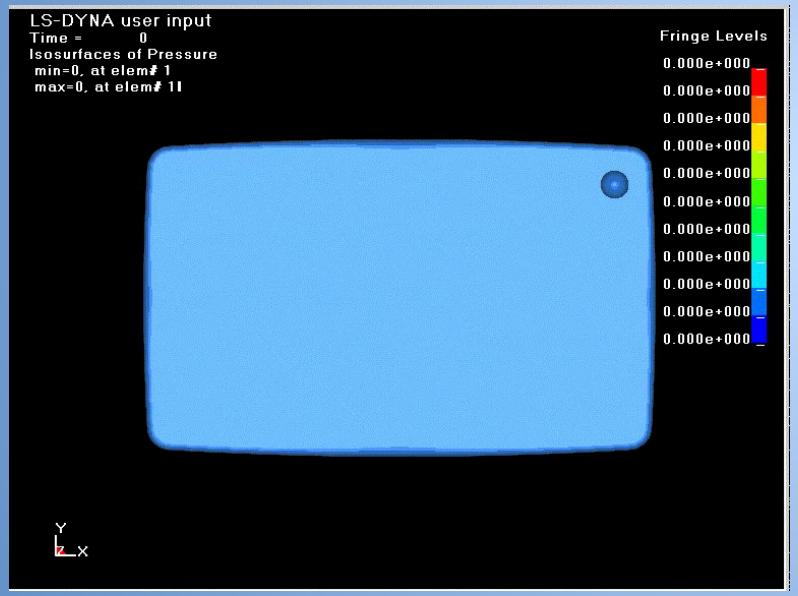
Spring Mass Model with Damper

- Low Computational Overhead
- Low Resolution
- Doesn't Handle Fracture

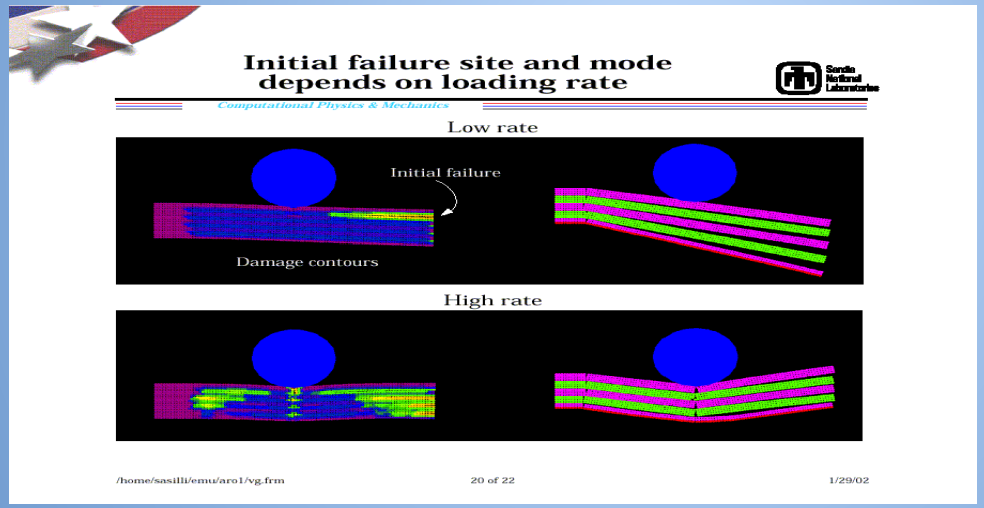


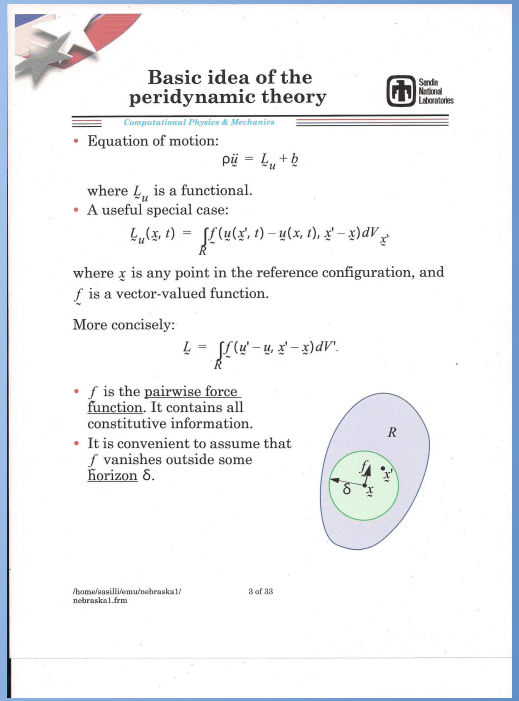
Two nodes as mass points connected by a parallel circuit of a spring and a damper.

LS-Dyna Model of UL-3952 3 **F**oot **P**ound Impact **Test**



EMU Simulations





Cloth Simulations

- Two dimensional elastic membranes
- Force based
- Positional

Collision Detection Issues

- Realistic interaction with environment
- Self intersections
- Techniques
  - Discrete a-posteriori
  - Continuous a priori
- Collisions with Environment
  - Well defined interior exists
  - Well defined interior does not exist
- Collisions between two Cloths (computationally complex)

Managing **Computational** Complexity

- Bounding Volumes
- Grids
- Coherence-Exploiting Schemes
- Hybrid Methods

Support **for** Soft Body Physics

https://youtu.be/KppTmsNFneg

- Digital Molecular Matter (DMM)
- Maya nCloth
- Physics Abstraction Layer (PAL)
- CryEngine www.youtube.com/watch?v=hmaHj6mpT0k
- EtXUBQ

Special Relativity

What is Special Relativity?

Well first, what is relativity?

"The theory that deals with motion of objects when their speed is close to the speed of light"

What is Special Relativity?

Special Relativity deals with the theories of Relativity when the objects being compared are moving uniformly

- This means no acceleration or rotation

Some things we need to talk about

Inertial Frames
The Galilei Transformation
The Lorentz Transformation

Inertial Frames

Remember Newton's First Law?

An inertial frame is a *frame of reference* where the *law of inertia* holds

non-inertial frames are those *frames of reference* that are *accelerating* in respect to the *inertial frames*

*For special relativity, we only care about inertial frames

The Galilei Transformation

Suppose we have two inertial frames moving in the *x* direction relative to one another at a constant velocity

Let's call them A: (x, t) and B: (x', t')

The Galilei Transformation provides us a way to translate between the two inertial frames

$$x' = x - vt \qquad t = t'$$

The Lorentz Transformation

In 1905, Einstein's theory of special relativity claimed that the Galilei Transformation is wrong at speeds closer to the speed of light.

Instead, Lorentz Transformation looks more like this:

$$x' = \frac{x - vt}{\sqrt{1 - \frac{v^2}{c^2}}}, \qquad t' = \frac{t - \frac{v}{c^2}x}{\sqrt{1 - \frac{v^2}{c^2}}}.$$

*Not as crazy as they look

Final notes of some importance

and things I just generally found interesting

One can not travel faster than the speed of light

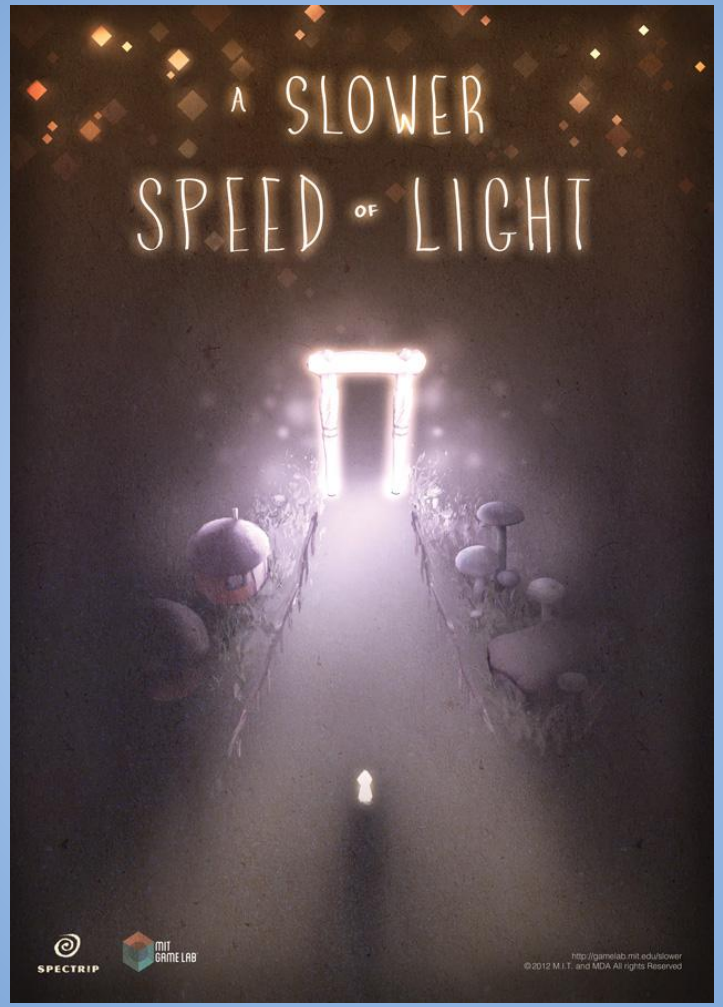
- As far as we know, this is not a matter of not having the technology, but rather *causality*.

The speed of light is always constant regardless of what inertial frame we are in.

What to do with special relativity?

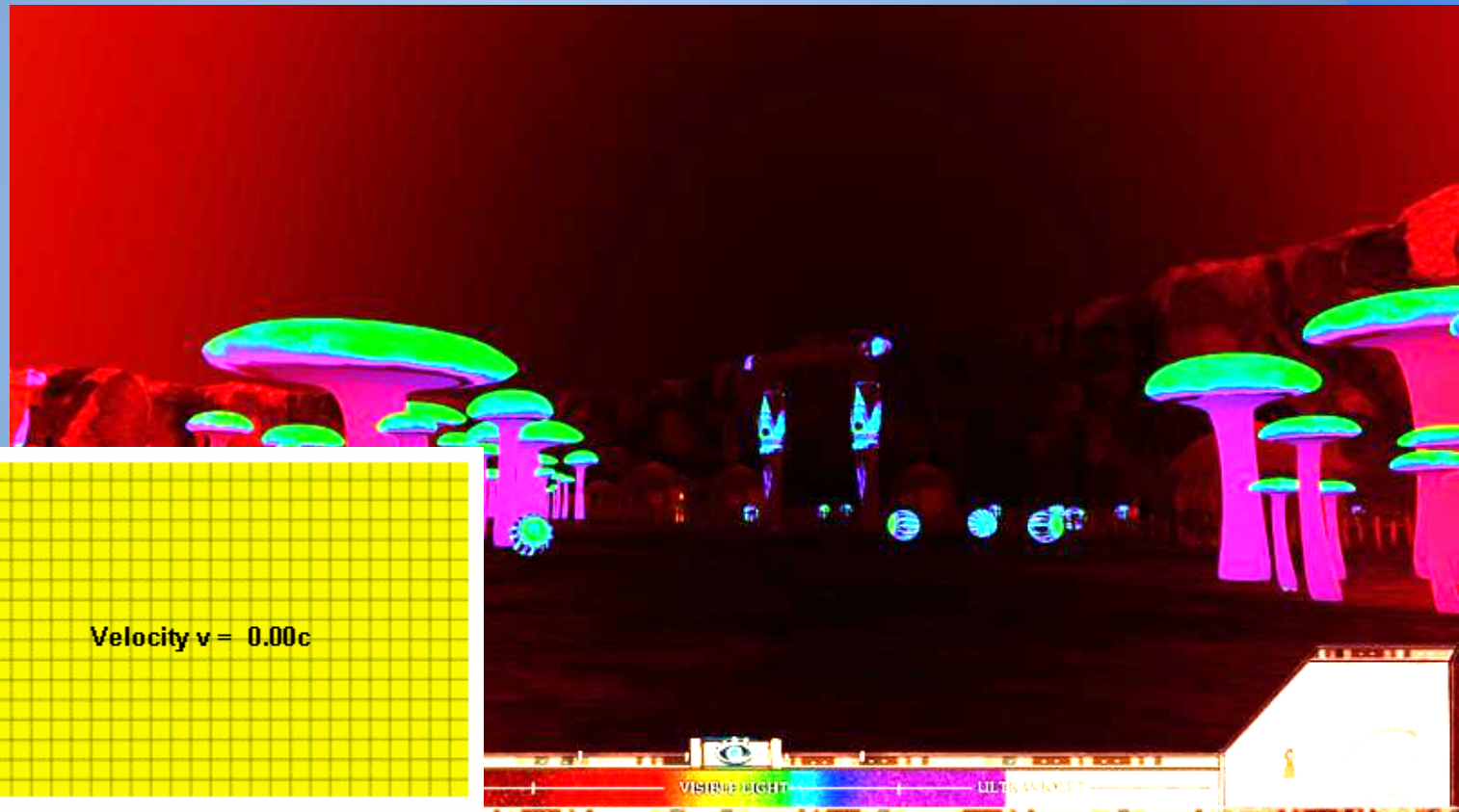
In 2012, the MIT Game Lab created a game called *A Slower Speed of Light*.

The game, although simple in design, allowed players to view the effects of special relativity first-hand as they collected orbs which incrementally lowered the speed of light.

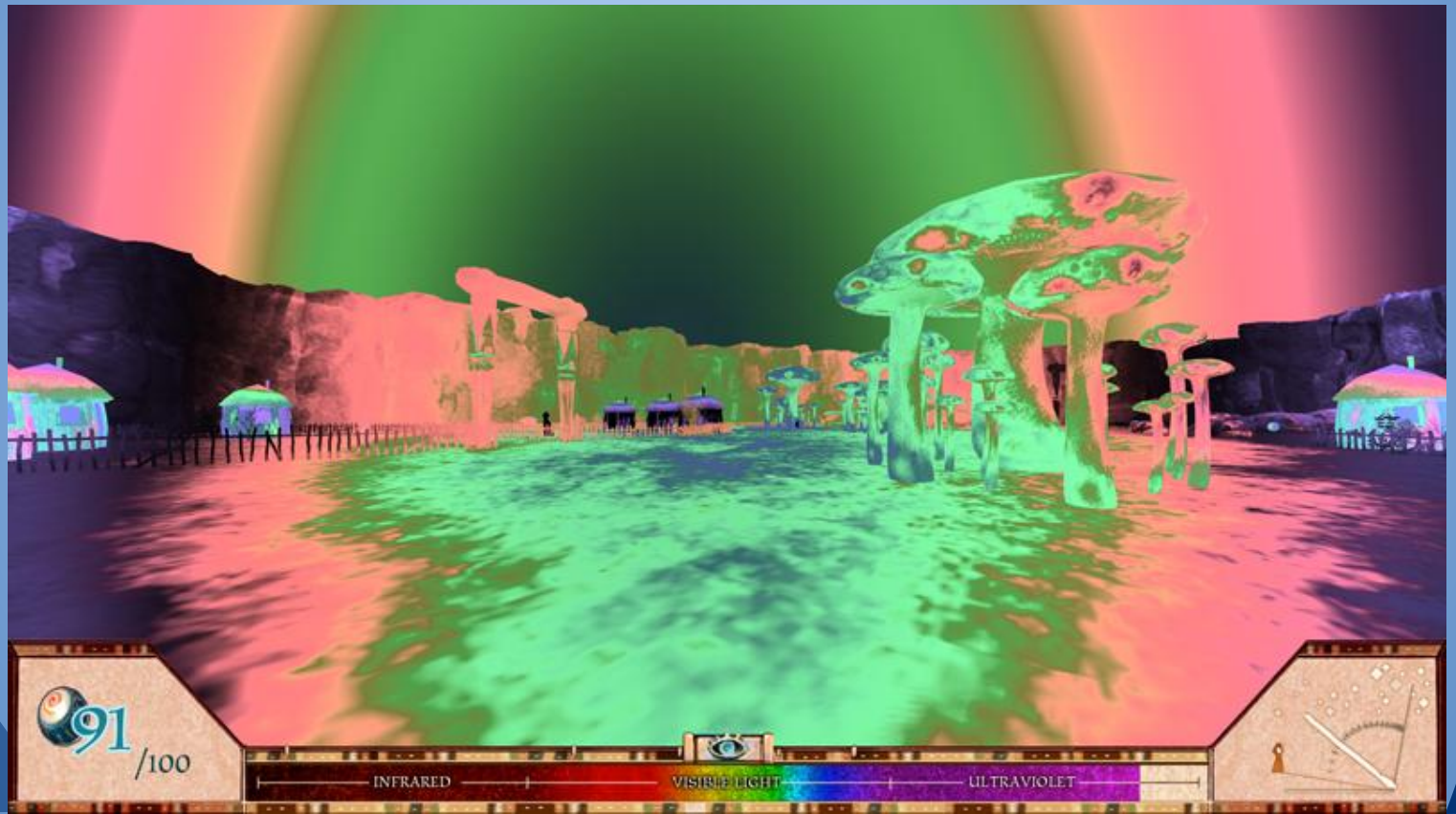

A SLOWER
SPEED OF LIGHT
SPECTRIP
MIT GAME LAB
http://gamelab.mit.edu/slower
©2012 M.I.T. and MDA All rights Reserved

A Slower Speed of Light

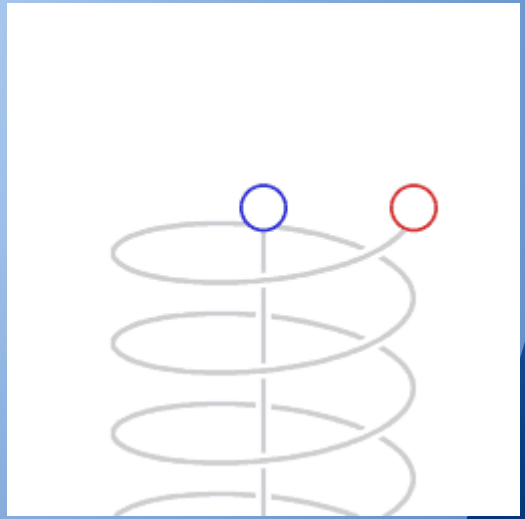
A Slower Speed of Light *Doppler Effect*



A Slower Speed of Light *"Searchlight Effect"*



A Slower Speed of Light *Time Dilation*

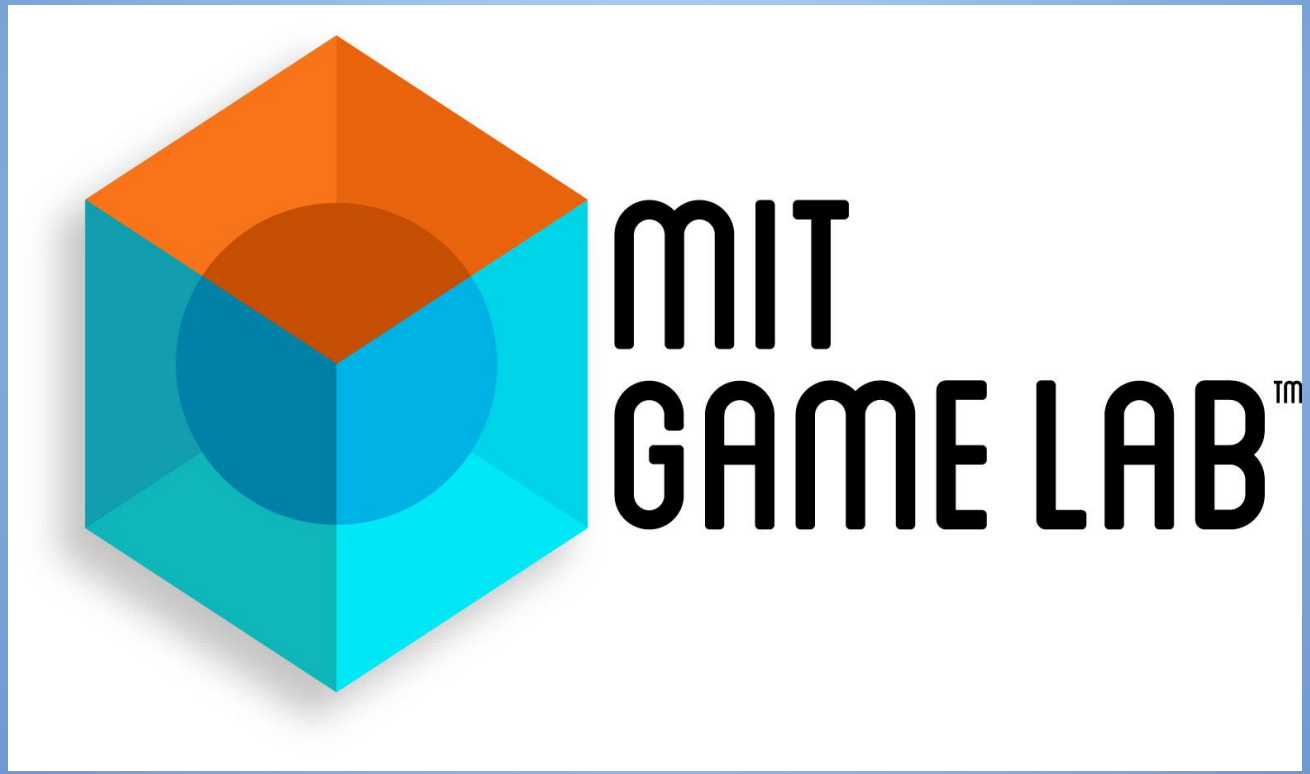


Open Relativity

With the game, MIT Game Lab also released a Unity tool-kit called OpenRelativity, which gave developers access to tools which could simulate Special Relativity in their own games

A demo has been created in the tool-kit to allow you to visualize its potential

Open Relativity -- Demo



Code Snippets -- Open Relativity

A great deal of the code is very complex, and harder to show and explain.

I will cover some examples that are a bit easier to sample, but most of the knowledge comes from looking at the theory, as the code is a reflection of it.

Code Snippets -- Open Relativity

Starting pretty basic… *Time Dilation*

```
 /*****************************
 * PART TWO OF ALGORITHM
 * THE NEXT 4 LINES OF CODE FIND
 * THE TIME PASSED IN WORLD FRAME
 * ***************************/
//find this constant
sqrtOneMinusVSquaredCWDividedByCSquared = (double)Math.Sqrt(1 - Math.Pow(playerVelocity, 2) / cSqrd);

//Set by Unity, time since last update
deltaTimePlayer = (double)Time.deltaTime;
//Get the total time passed of the player and world for display purposes
if (keyHit)
{
    totalTimePlayer += deltaTimePlayer;
    if (!double.IsNaN(sqrtOneMinusVSquaredCWDividedByCSquared))
    {
        //Get the delta time passed for the world, changed by relativistic effects
        deltaTimeWorld = deltaTimePlayer / sqrtOneMinusVSquaredCWDividedByCSquared;
        //and get the total time passed in the world
        totalTimeWorld += deltaTimeWorld;
    }
}
```

Code Snippets -- Open Relativity

The shader gets pretty crazy… *Lorentz Transform*

```
//get the new position offset, based on the new time we just found
//Should only be in the Z direction

riw.x += rotateViw.x * tisw;
riw.y += rotateViw.y * tisw;
riw.z += rotateViw.z * tisw;

//Apply Lorentz transform
// float newz =(riw.z + state.PlayerVelocity * tisw) / state.SqrtOneMinusVSquaredCWDividedByCSquared;
//I had to break it up into steps, unity was getting order of operations wrong.
float newz = (((float)speed*_spdOfLight) * tisw);

newz = riw.z + newz;
newz /= (float)sqrt(1 - (speed*speed));
riw.z = newz;
if (speed != 0)
{
    float trx = riw.x;
    float trry = riw.y;

    riw.x = riw.x * (ca + ux*ux*(1-ca)) + riw.y*(ux*uy*(1-ca)) - riw.z*(uy*sa);
    riw.y = trx * (uy*ux*(1-ca)) + riw.y * ( ca + uy*uy*(1-ca)) + riw.z*(ux*sa);
    riw.z = trx * (uy*sa) - trry * (ux*sa) + riw.z*(ca);
}
```

But again… why?

So how exactly can you use this?

Although the scope of special relativity has limits, it still has plenty of potential applications in the gaming world.

- Creating a racing game at near speed of light speeds
- Enhancing visual effects
- Giving the illusion of speed

Questions?

Obligatory Physics GIF