Praktikum: Echtzeit Computergrafik



tum.3D

computer graphics & visualization

Assignment 10

Assignment 10

- Task overview:

1. Configuring your guns
2. Spawning and updating projectiles
3. Rendering projectiles
4. Collision detection and enemy death

Assignment 10

- Task overview:

1. Configuring your guns
2. Spawning and updating projectiles
3. Rendering projectiles
4. Collision detection and enemy death

Assignment 10

- Notice we have two guns?
- Each gun should be able to fire separately
- Each gun should have unique properties
  - Fire rate, damage, projectile speed, visuals…
  - As always: configurable!

Configuration Changes

- Configuration examples
  - Guns:

```
22  # Gatling/PlasmaGun spawn_pos_view speed gravity cooldown damage sprite_tex_index sprite_radius
23  GatlingGun -2 0.2 6.6 500 10 0.1   10 0 1
24  PlasmaGun   2 0.2 5.5  50  0 0.75 100 1 1.5
```

  - Projectile sprites:

```
1  # Sprite tex_path
2  Sprite parTrailGatlingDiffuse.dds
3  Sprite parTrailPlasmaDiffuse.dds
```

- As always: Encapsulate those parameters and parse the config file
  - Only two predefined guns need to be supported

Gun Behaviour

- Reminder: Callbacks
  - This should seem familiar

```
1155 void CALLBACK OnKeyboard( UINT nChar, bool bKeyDown, bool bAltDown, void* pUserContext )
```

  - Is called when a key is pressed or released
  - So for each gun…
    - Check for a specific key (select whatever you want)
    - Store the last state „somewhere“

Gun Behaviour

- In each frame (OnFrameMove()):

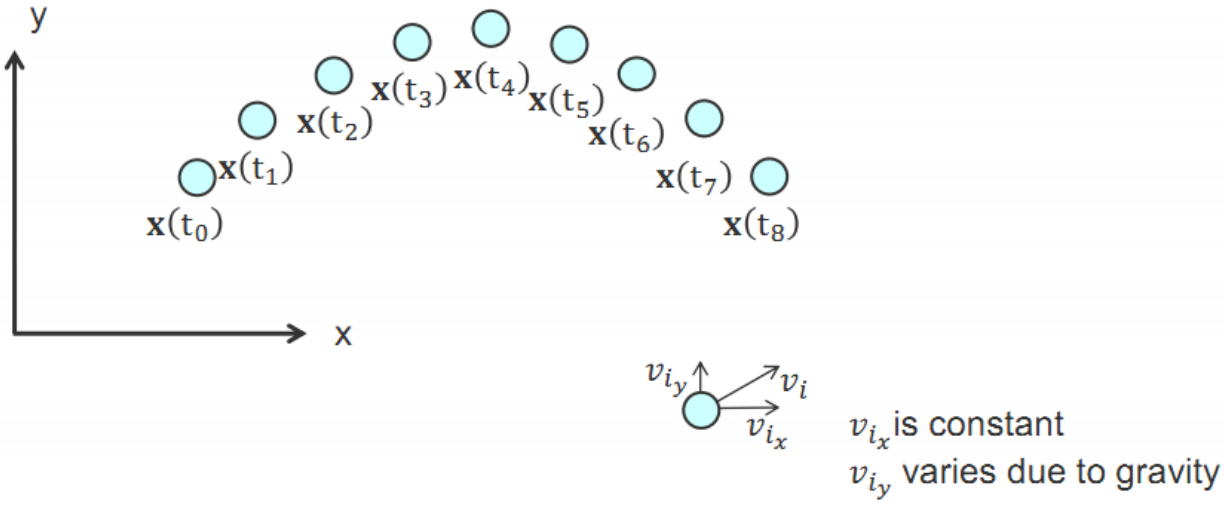
Spawning Projectiles

- Add projectile to a global container
  (`std::vector` / `std::list`)

- Initialize projectile settings
  - Direction: `g_Camera.GetWorldAhead()`
  - Speed: Gun parameters!
  - Position: Gun parameters… but…
    - Given in view space ➔ Transform to world space first
    - Apply inverse view transform – you can get this directly from `g_Camera.GetWorldMatrix()`

Updating Projectiles

- Review the lecture slides on projectile motion!

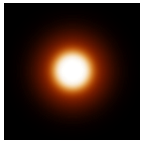


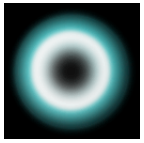
- Forward Euler integration is sufficient here:

$$\vec{v}_{new} = \vec{v}_{old} + \vec{g}_{gun} \cdot \Delta t$$

$$\vec{p}_{new} = \vec{p}_{old} + \vec{v}_{new} \cdot \Delta t$$

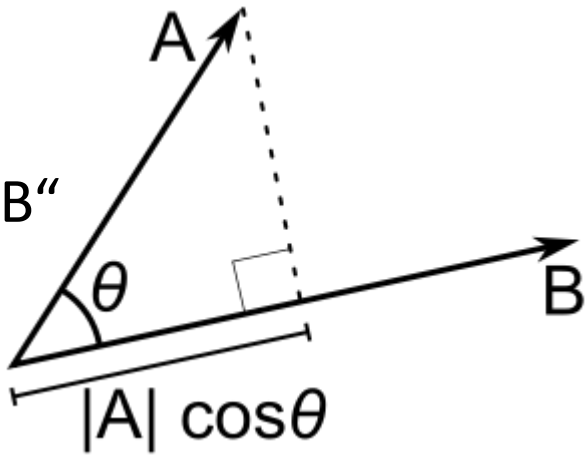
Rendering Projectiles





- Use your existing sprite rendering methods
  - Each gun defines sprite size and texture
  - Pass positions of all projectiles to your sprite renderer
- Order is important ➔ Sort before rendering

- Particle depth in view space: dot product of position and view direction
  - dot(A, B): |A|*|B|*cos(A,B)
  - dot(A, B): Length of „A projected onto B“
  - B must be unit vector
  - View direction: camera class!



Rendering Projectiles

- Reminder from Assignment 1: sorting
  - Use `std::sort` instead of your own sorting algorithm
    - http://www.cplusplus.com/reference/algorithm/sort/

function template
sort <algorithm>

```
template <class RandomAccessIterator>
  void sort ( RandomAccessIterator first, RandomAccessIterator last );

template <class RandomAccessIterator, class Compare>
  void sort ( RandomAccessIterator first, RandomAccessIterator last, Compare comp );
```

  - Iterators: From your container!
  - Compare:
    - Function

```
std::vector<int> myvector;
bool myfunction(const int& i, const int& j) { return (i < j); }

std::sort(myvector.begin(), myvector.end(), myfunction);
```

    - Function object / lambda expression

Rendering Projectiles

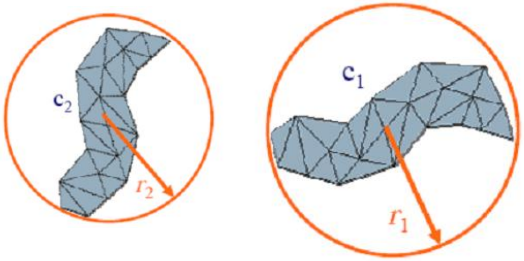
- `std::sort` checks for „strict ordering“!
  - Precision problems if you directly calculate the dot product in the comparator!

```
dot(v1, v) < dot(v2, v) == true
dot(v2, v) < dot(v1, v) == true
```

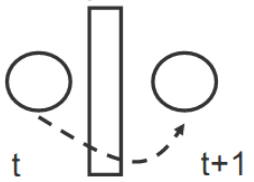
- Solution: Precompute camera distance for all particles

Collisions With Enemies

- Broad phase collision only
- Each enemy type defines a bounding sphere size



- Our projectiles are spheres, so:
  - Simple sphere / sphere collision
  - Discrete collision detection is sufficient
  - … so make sure your projectiles aren‘t moving too fast

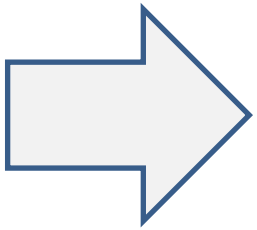


- Reminder from the lecture slides: no intersection if

$$(c_1 - c_2)^T (c_1 - c_2) > (r_1 + r_2)^2$$

Collisions With Enemies

- On hit:
  - Delete projectile
  - Decrease enemy hitpoints
  - Delete the enemy ship if hitpoints reach 0



Questions?