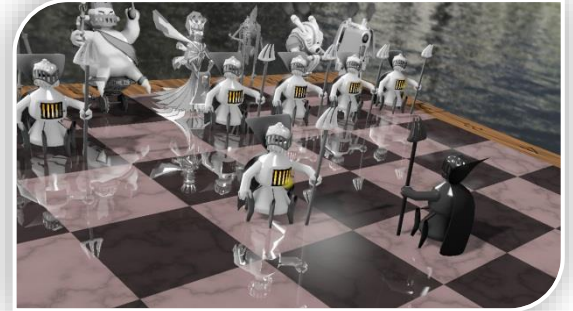
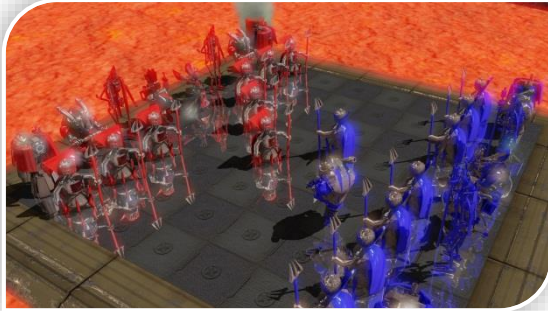


Praktikum: Echtzeit Computergrafik



tum.3D
computer graphics & visualization

Assignment 8

- This week: Enemy ships!



- Simple enemy model
 - Enemies spawn in fixed intervals
 - Enemies fly along straight paths with constant velocity
 - Paths are randomly generated
- Enemy Types & Enemy Instances
 - Enemy Types are defined in config file
 - Name, hitpoints, speed, ...
 - Mesh + transformation
 - Enemy Instances are generated at runtime
 - Enemy Type name
 - Position + velocity
 - Remaining hitpoints, ...



- Example definition in config file

```
Mesh Amy ...
```

```
#EnemyType name hitpoints size speed mesh_name scale rot_x _y _z trans_x _y _z  
EnemyType AmyShip 100 10 50 Amy 0.5 0 180 0 0 0 0
```

- Example internal representation (similar to CockpitObjects and GroundObjects)

```
struct EnemyType  
{  
    int hitpoints;  
    ...  
};  
  
//Dictionary of EnemyTypes  
std::map<std::string,EnemyType> enemyTypes;
```

- Example internal representation

```
struct EnemyInstance
{
    ...
    XMVECTOR pos;    //position p in world space
    XMVECTOR vel;    //velocity v in world space
    ...
};

std::list<EnemyInstance> g_EnemyInstances;
```

- Store instances in a list
 - `std::list` → doubly-linked list in C++
 - Allows for efficient removal of arbitrary elements
 - Usage very similar to `std::vector`

- A list of `int`

```
std::list<int> l;  
  
//insert elements at front or end of list  
l.push_back(13);  
l.push_front(5);  
  
//iterate over all list elements  
for (auto it=l.begin(); it!=l.end(); it++)  
{  
    //you have to use a reference ("int&") in order to affect "value" in list  
    int value_copy = *it;  
    int& value      = *it;  
    //do something with value...  
}  
  
//remove all elements of list  
l.clear();
```

- Removing single elements requires caution!

- Problem: iterate over list and remove certain elements
- Manual `erase()`

```
//remove all elements that are equal to 5
for (auto it=l.begin(); it!=l.end(); it++)
{
    int value = *it;
    if (value == 5) {
        l.erase(it); //WRONG! call to erase invalidates "it"
    }
}
```

WRONG!

- Problem: iterate over list and remove certain elements
- Manual `erase()`

```
//remove all elements that are equal to 5
for (auto it=l.begin(); it!=l.end(); ) //no it++ in loop-header!
{
    int value = *it;
    if (value == 5) {
        auto it_remove = it;
        it++; //increment it before call to erase
        l.erase(it_remove);
    }
    else { it++; }
}
```

- Alternative: `remove_if()` method of `std::list`
 - Using a predicate function or class
http://www.cplusplus.com/reference/stl/list/remove_if/
 - Using lambda expressions (advanced)
<http://msdn.microsoft.com/de-de/library/dd293608>

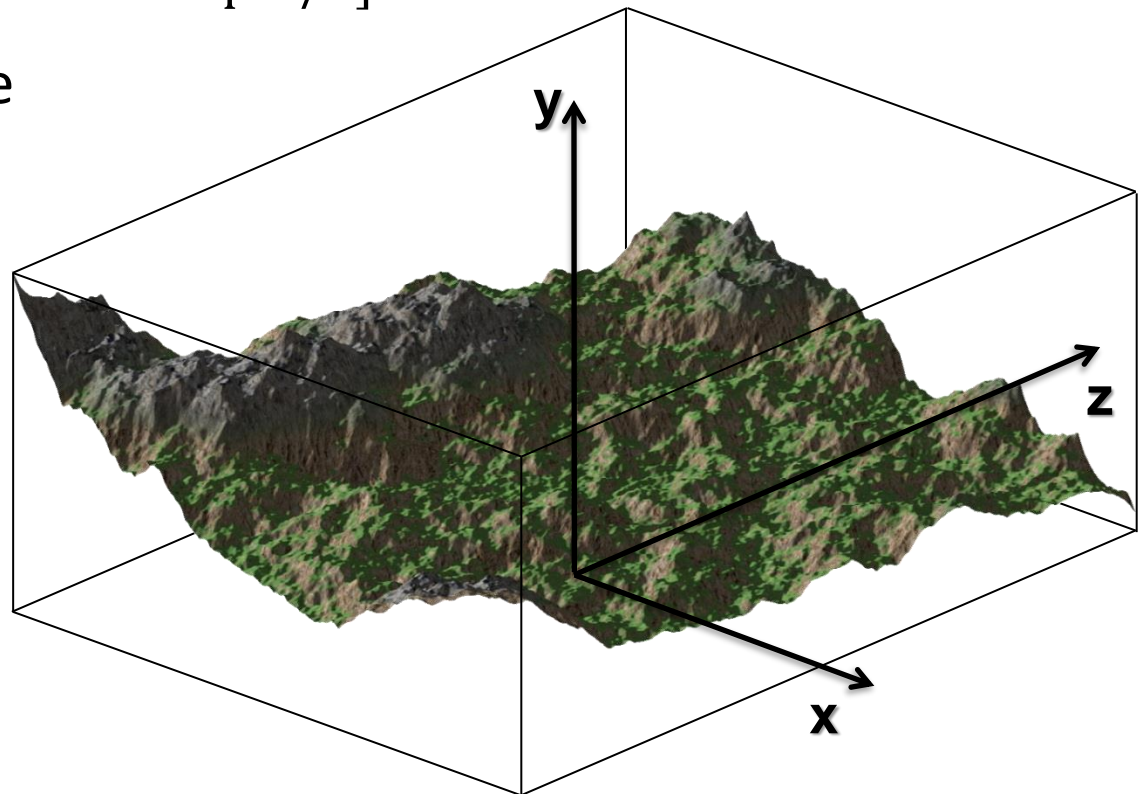
- Problem: Spawn enemies every g_SpawnInterval seconds
 - Add this to your config
- Spawn enemies in OnFrameMove()
 - Called every frame before OnD3D11FrameRender()
 - fElapsedTime contains time since last frame (in seconds)
 - Use global timer variable

```
float g_SpawnTimer    = 0.0f;

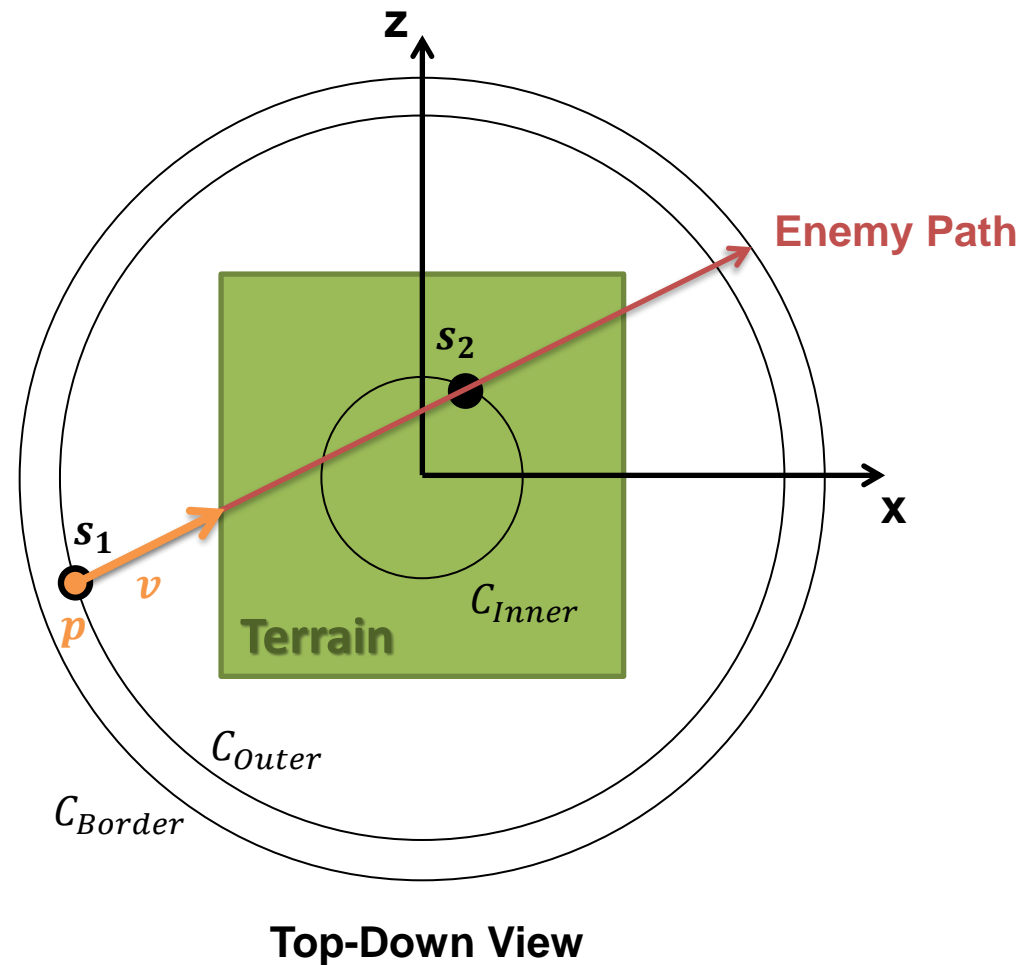
void CALLBACK OnFrameMove( ..., float fElapsedTime, ... )
{
    ...
    g_SpawnTimer -= fElapsedTime;

    if (g_SpawnTimer<0)
    {
        g_SpawnTimer += g_configParser.GetSpawnInterval();
        //spawn new enemy here
    }
    ...
}
```

- Position and velocity of enemies is stored in world space
- Terrain range of values
 - $x \in [-\text{TerrainWidth}/2; \text{TerrainWidth}/2]$
 - $y \in [0; \text{TerrainHeight}]$
 - $z \in [-\text{TerrainDepth}/2; \text{TerrainDepth}/2]$
- Of course, enemies are not restricted to this range of values



- Example enemy spawn model



- Example enemy spawn model
 - Config file parameters: h_{min} , h_{max} , $speed$
 - $\mathbf{s}_1.xz := \text{random point on circle } C_{outer}$
 - $\mathbf{s}_2.xz := \text{random point on circle } C_{inner}$
 - $\mathbf{s}_1.y := \mathbf{s}_2.y := \text{random height} \in [h_{min}; h_{max}]$
 - Enemy position $\mathbf{p} := \mathbf{s}_1$
 - Enemy velocity $\mathbf{v} := speed \cdot \text{normalize}(\mathbf{s}_2 - \mathbf{s}_1)$
 - Remove enemy if outside C_{border}
- Hint: Random point \mathbf{x} on 2D circle with radius r
 - $\alpha := \text{random number} \in [0; 2\pi]$
 - $\mathbf{x} := r \cdot (\sin(\alpha), \cos(\alpha))$

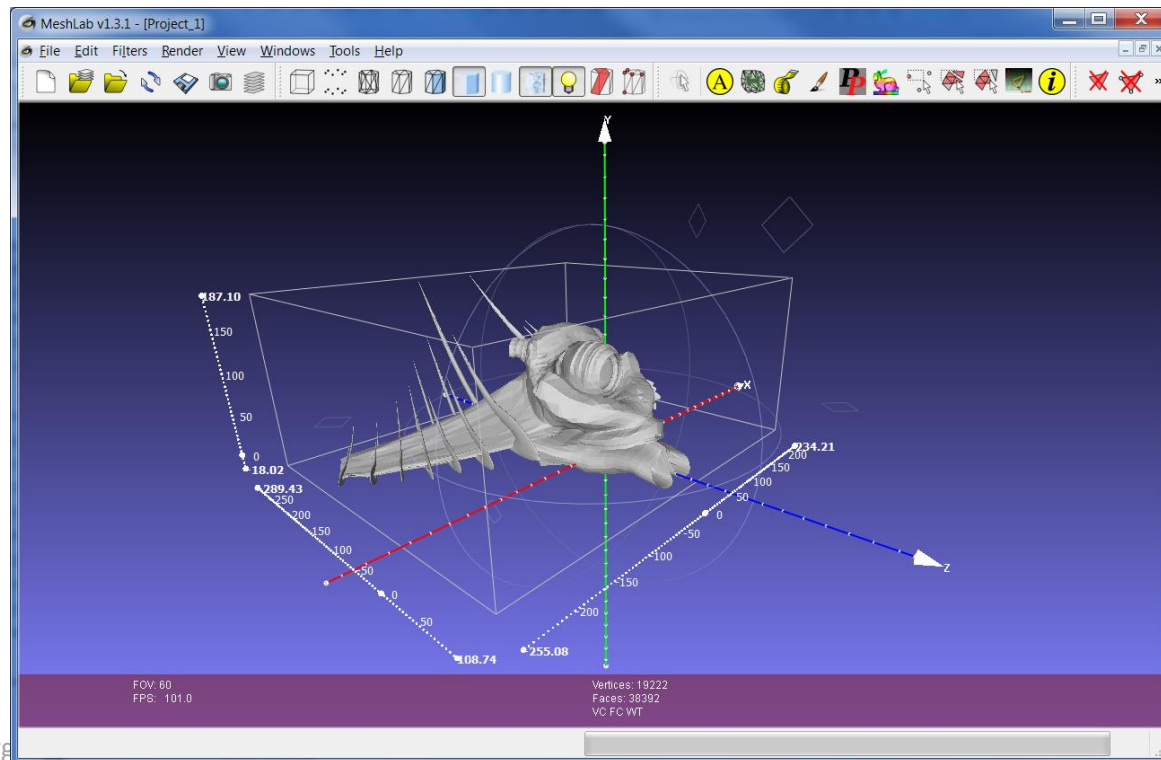
(trigonometry in C++: `XM_PI`, `std::sin()`, `std::cos()`, ...)

- Each enemy instance has a
 - Position $\mathbf{p} \in R^3$
 - Velocity $\mathbf{v} \in R^3$
- Position update in OnFrameMove()
$$\mathbf{p} := \mathbf{p} + h \cdot \mathbf{v} \quad (h = fElapsedTime)$$
- This is a simple case of projectile motion!
(with no gravity and constant velocity)!

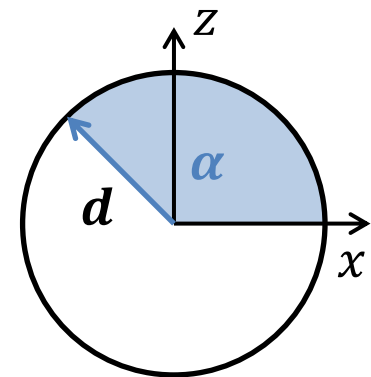


- Cockpit Objects: $M_{WorldView} = M_{Obj} \cdot M_{CamWorld} \cdot M_{CamView} (\rightarrow \text{Ass. 6})$
- Ground Objects: $M_{WorldView} = M_{Obj} \cdot M_{CamView} (\rightarrow \text{Ass. 7})$
- Enemy Instances: $M_{WorldView} = \underbrace{M_{Obj} \cdot M_{Anim}}_{M_{World}} \cdot M_{CamView}$
- M_{Obj} is given by the config file (see slides for Assignment 07!)
- $M_{CamView}$ is the camera view matrix
- M_{Anim} : rotation/translation according to enemy position/velocity

- Object space correction: $M_{Scale} \cdot M_{RotX} \cdot M_{RotY} \cdot M_{RotZ} \cdot M_{Trans}$
- Unify pose of enemy meshes
 - Scale: World space size of enemy ship
 - Rotation: $+x \leftrightarrow \text{front}$, $+y \leftrightarrow \text{up}$
 - Translation: Ship mesh (roughly) centered at origin (that's already the case)



- M_{Anim} : Move enemy mesh to position \mathbf{p} , facing into direction $\mathbf{d} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$
 - Rotation + translation
 - Assumption: Ships are always aligned parallel to the ground
- Corresponding matrix:
 - M_{Anim} = Rotation around y-Axis with angle α
... followed by ...
translation by \mathbf{p}
 - Calculate α using `std::atan2()` with $d.x$ and $d.z$



Questions?

