Praktikum: Echtzeit Computergrafik



tum.3D

computer graphics & visualization

Assignment 7

- This week: From one mesh to multiple objects

Assignment 7

- Problem: Manage multiple objects
  - Stationary objects on the ground
  - „Cockpit“ objects attached to the camera
  - Some objects should have same appearance, but different positioning (→ do not load the same mesh twice)
- Idea: Separate meshes and objects
  - Mesh
    - Geometry & Textures
  - Object
    - Reference to a mesh
    - Type: ground / cockpit (more later…)
    - Position: Scale / rotation / translation

Assignment 7

- Tasks
  - Generate resources for new meshes
  - Meshes & objects:
    - Define in config file
    - Parse to internal representation
    - Manage using appropriate data structures
    - Render (using your knowledge and existing shaders from Ass. 5)
- C++ only
- More freedom – less Copy&Paste – shorter assignment (many solution ideas on the following slides)

Config File

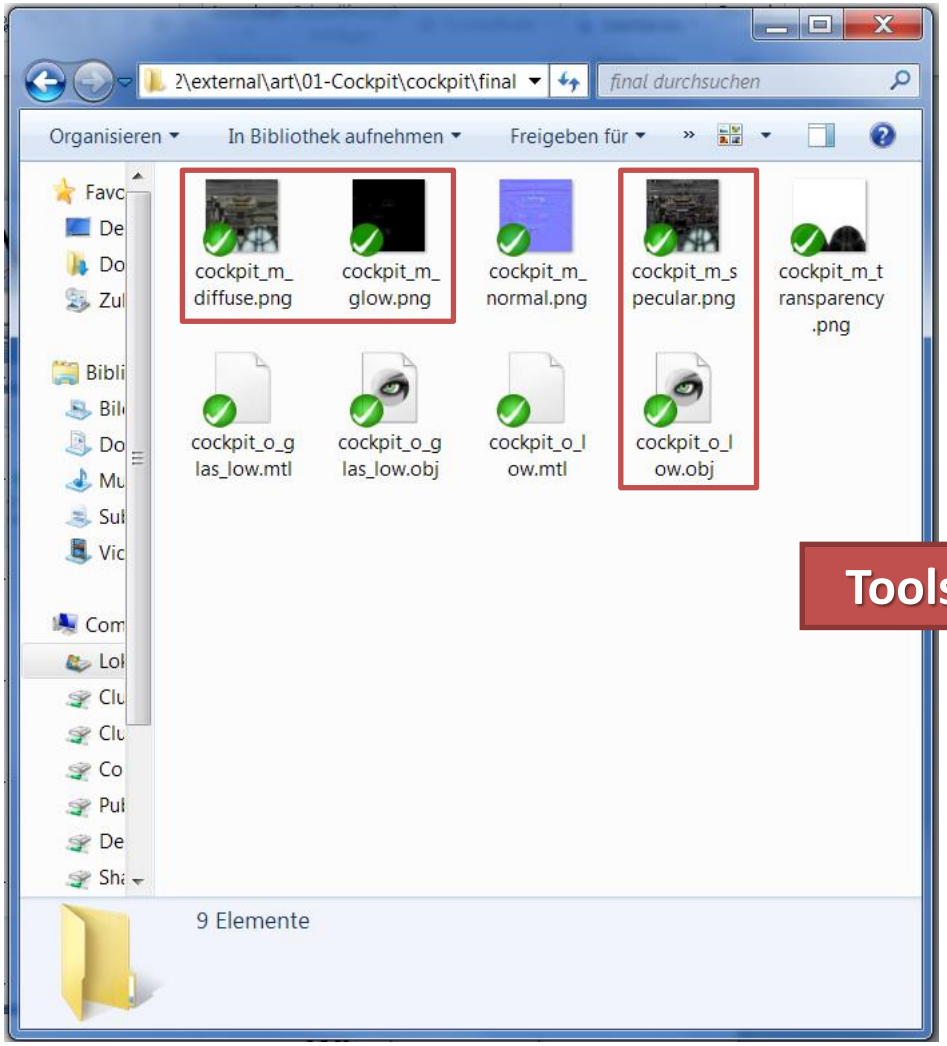
- Definition of meshes and objects in game.cfg, e.g.

```
# Mesh name t3d_path diffuse_path specular_path glow_path
Mesh Cockpit cockpit_o_low.t3d cockpit_m_diffuse.dds cockpit_m_specular.dds cockpit_m_glow
.dds
Mesh Barracks barracks.t3d barracks_diffuse.dds barracks_specular.dds -
Mesh Tower tower.t3d tower_diffuse.dds tower_specular.dds -

# CockpitObject mesh_name scale rot_x rot_y rot_z trans_x trans_y trans_z
CockpitObject Cockpit 0.5 0 90 0 0 -0.8 2.1
CockpitObject GatlingGunBase 1 0 90 0 1 0.5 2

# GroundObject mesh_name scale rot_x rot_y rot_z trans_x trans_y trans_z
GroundObject Barracks 1 0 15 0 100 0 0
GroundObject Barracks 0.6 0 0 0 -16 0 22
GroundObject Tower 0.8 0 30 0 -10 0 15
```

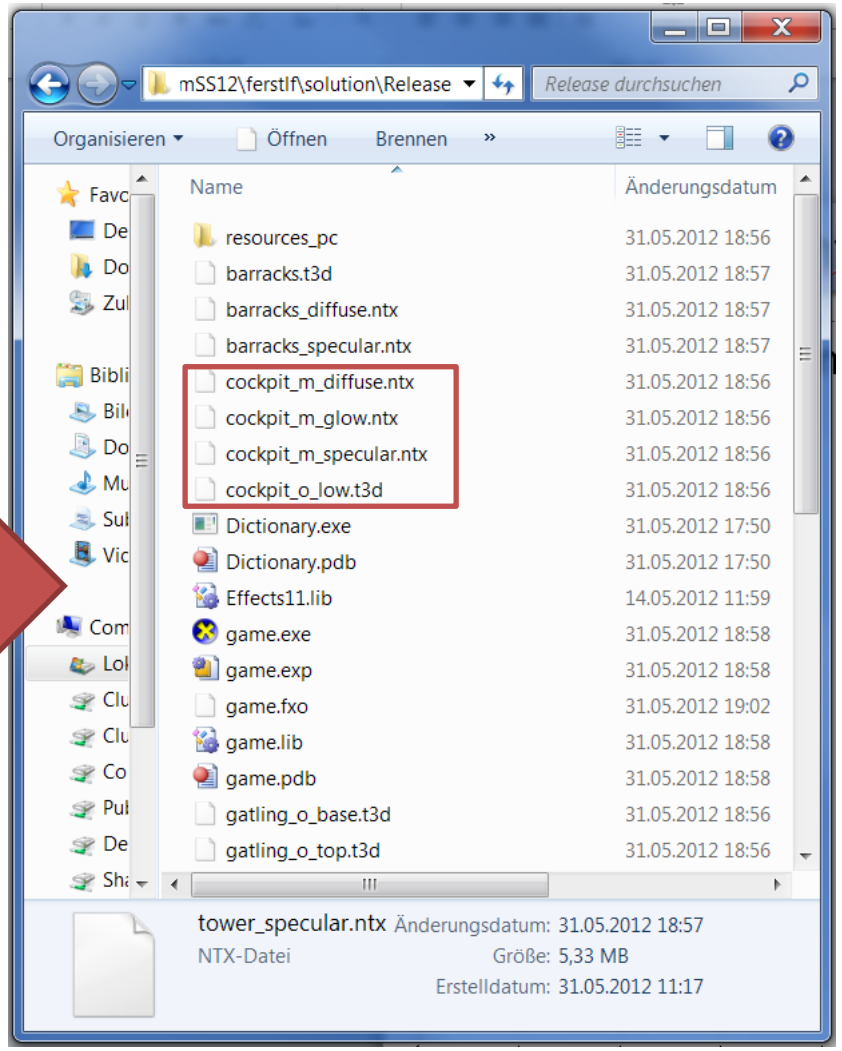
  - Note: Values are just examples here!

- Parse into your own internal representation
  - Mesh: Dictionary of meshes identified through name-identifiers (use existing mesh class)
  - CockpitObject / GroundObject: Array(s)/List(s) of objects (create your own struct(s) / class(es))

Resources



Tools



What we have

(authoring format: *.png, *.obj)

What we need

(runtime format: *.dds, *.t3d)

Automated Resource Generation

- Add everything to the command line of your ResourceGenerator
  - E.g. for cockpit mesh (from Ass. 6):

```
"$(SolutionDir)..\..\external\Tools\bin\obj2t3d.exe" -i "$(SolutionDir)..\..\external\art\01-Cockpit\cockpit\final\cockpit_o_low.obj" -o "$(OutDir)resources\cockpit_o_low.t3d" –y

"$(OutDir)texconv" -o "$(OutDir)resources" -srgbi -f BC1_UNORM_SRGB "$(SolutionDir)..\..\external\art\01-Cockpit\cockpit\final\cockpit_m_diffuse.png"

"$(OutDir)texconv" -o "$(OutDir)resources" -srgbi -f BC1_UNORM_SRGB "$(SolutionDir)..\..\external\art\01-Cockpit\cockpit\final\cockpit_m_specular.png"

"$(OutDir)texconv" -o "$(OutDir)resources" -srgbi -f BC1_UNORM_SRGB "$(SolutionDir)..\..\external\art\01-Cockpit\cockpit\final\cockpit_m_glow.png"
```

  - Copy & paste existing lines
  - Adjust **input** and **output**

Parsing Tipps

- Comments & Simple Error Checking

```
while(!stream.eof())
{
    stream >> var;

    // Comments: skip lines starting with a #
    if(var.empty() || var[0] == '#')
    {
        std::string comment;
        std::getline(stream, comment);
        continue;
    }

    ...

    // simple error check
    if(stream.fail() && !stream.eof()) {
        MessageBoxA (NULL, ("Warning: Failed parsing config variable "
            + var).c_str(), "Config file error", MB_ICONERROR | MB_OK);
        exit(-1);
    }
}
```

Dictionaries in C++

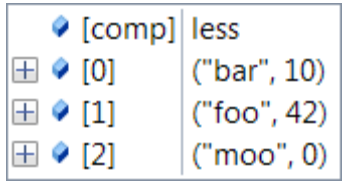
- Dictionaries / associative containers
  - Realize a mapping from keys (strings in our case) to values
  - E.g. „foo“ → 5
    „bar“ → 42
    „moo“ → 100
  - Keys are unique
  - Values can be accessed through keys
  - Key/value pairs can be inserted and deleted
- In C++: `std::map<key_type,value_type>`
  http://www.cplusplus.com/reference/map/map/
  - `key_type` has to support „<“-operator → std::string
  - `value_type` can be anything

C++ Dictionary - Simple Example

- Ensure that `<map>` and `<string>` are included

```
//create empty map
std::map<std::string,int> m;

//insert "foo"->42 and "bar"->10
m["foo"] = 42;
m["bar"] = 10;

//access "foo" and "bar"
int f = m["foo"], b = m["bar"];
std::cout << f << " " << b << "\n"; //output: 42 10

//Caution! the []-operator automatically inserts elements that do not exist
std::cout << m["moo"] << std::endl; //output: 0 (= the result of "int()" )
std::cout << m.size() << std::endl; //output: 3

//for safe accesses, check first whether element exists
if ( m.find("foo")==m.end() ) std::cout << "foo is not in the map" << std::endl;

//remove element
m.erase("foo");
```

Visual-Studio Debugger also supports std::map!



C++ Dictionary – More Operations

```
// Iterate over all elements
for (auto it = m.begin(); it != m.end(); ++it)
{
        std::string   key = it->first;
        int           value = it->second;
        // Do something with key and / or value...
}

// Remove all elements
m.clear();
```

- Your dictionary should store Pointers (`Mesh*`)

- You still need `new` and `delete` for objects in the map!
  - `erase()` and `clear()` will only remove the pointers, not free the memory they point to
  - `[]` will insert a new pointer if the key does not exits, but not allocate any memory
  - Iterate over all objects and call `delete` before clearing the map

Ground and Cockpit Objects

- Define your own types to mirror definitions in config file, e.g. a struct for cockpit objects

game.cfg

```
CockpitObject Cockpit 0.5 0
90 0 0 0.5 1.5
```

```
struct CockpitObject
{
    // your member variables here
};

// contains all ground objects
std::vector<GroundObject> groundObjects;
```

(Hint: `std::vector` has a method called `push_back()` for easy insertion of new elements at the back of the array)

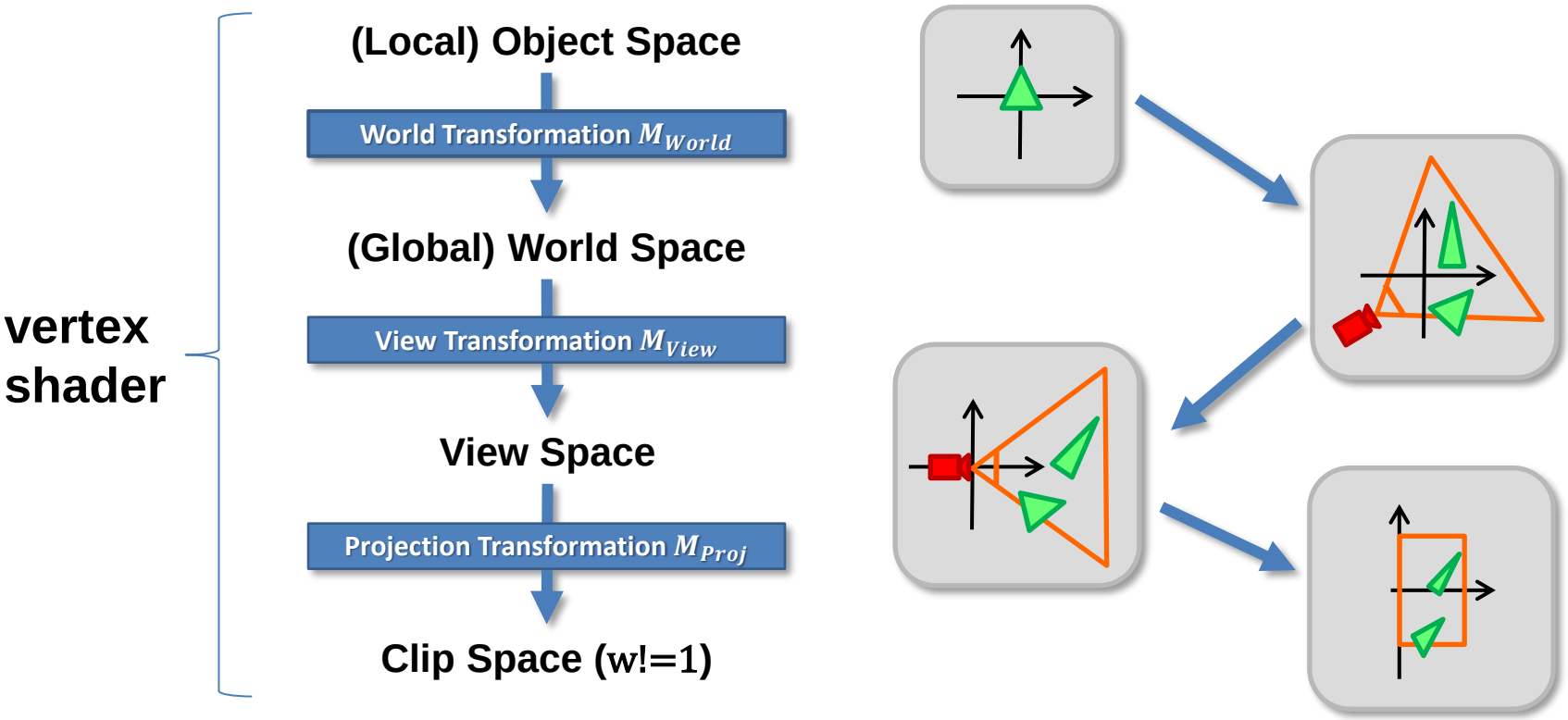
- Define a similar type for ground objects or extend the existing type to support both

Object Transformations

- For now: the only difference between ground and cockpit objects is the transformation for **rendering**
- $M_{Obj} = M_{Scale} \cdot M_{RotX} \cdot M_{RotY} \cdot M_{RotZ} \cdot M_{Trans}$
- Cockpit: $M_{WorldView} = \underbrace{M_{Obj} \cdot M_{CamWorld}}_{M_{World}} \cdot \underbrace{M_{CamView}}_{M_{View}}$
- Ground Objects: $M_{WorldView} = \underbrace{M_{Obj}}_{M_{World}} \cdot \underbrace{M_{CamView}}_{M_{View}}$
- $M_{Scale}, \dots, M_{Trans}$ are defined in config file
- $M_{CamView}$ is the camera view matrix, $M_{CamWorld}$ the camera world matrix

Ground Objects

- Transformations for ground objects



Questions?