GED Practical Course

tum.3D

computer graphics & visualization

Assignment 2

Goal:

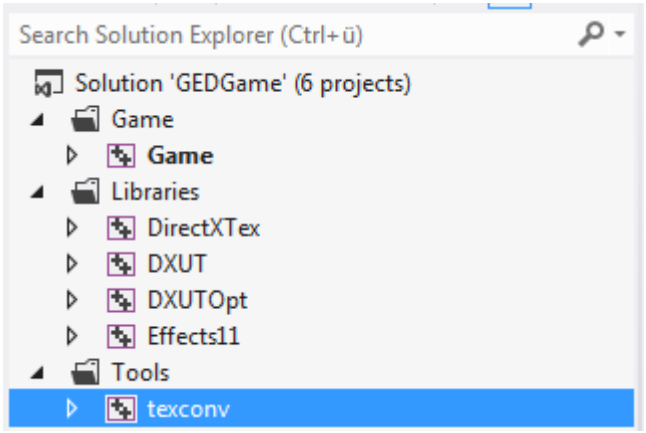
- Creating the height field of a fractal landscape
- Check the result in the provided height field viewer
- Refine until it looks satisfying

Preparations

- You will get a working solution from us
  - Contains several required libraries and a game template
  - We will get to the template in two weeks

- First task: Add your own project
  - Will require you to set up some include- and library directories
  - Step-by-step instructions in the assignment!

Solution contents



Game
Game template

DirectXTex
Texture loading utility library

DXUT / DXUTOpt
DirectX utility library

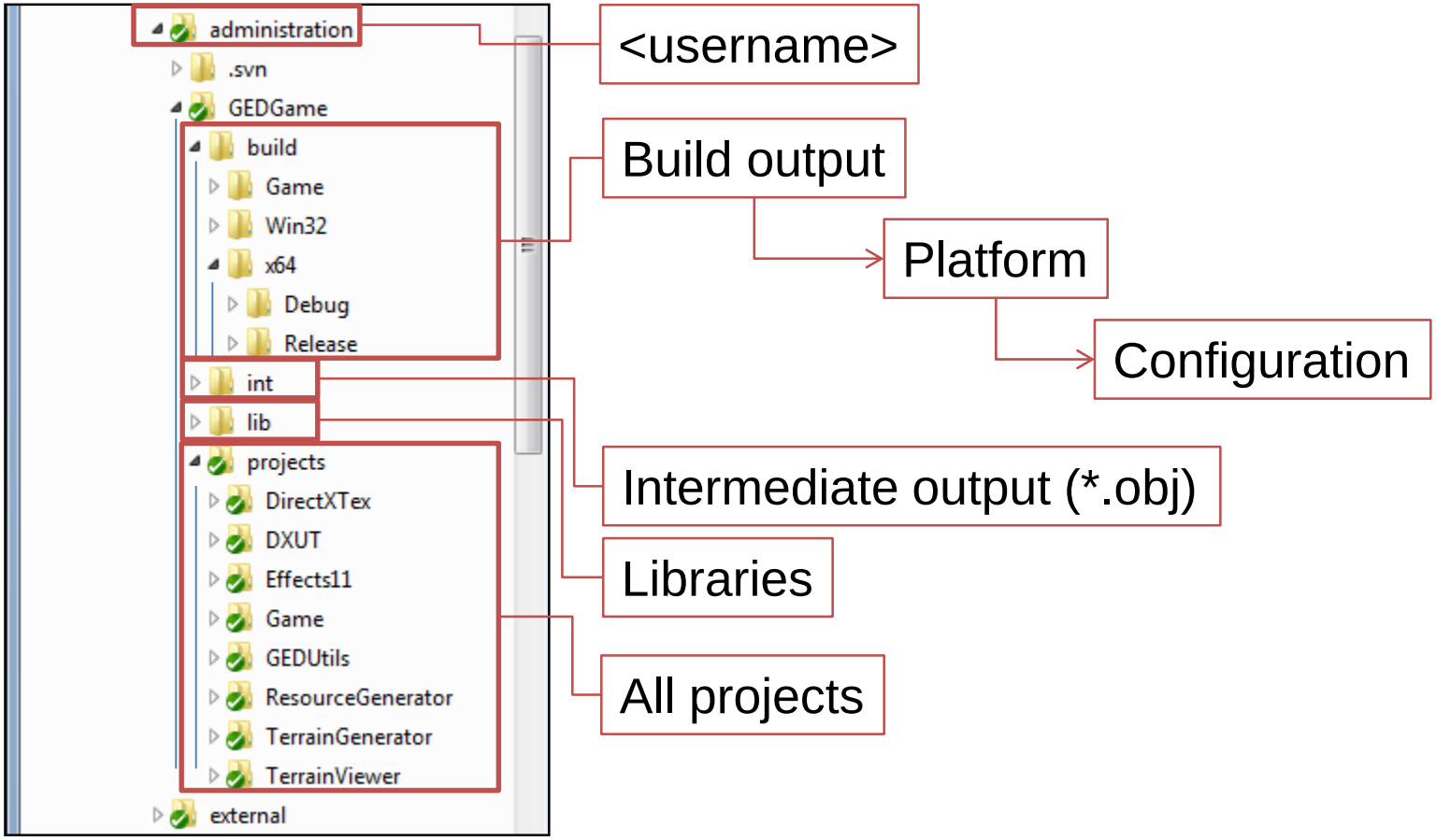
Effects11
Shader framework library

In addition:

GEDUtils
Some GED-specific utilities

Texconv
Texture processing command line tool

Solution / Build Structure



Paths

- Always use relative paths for include / library paths
  - E.g. external as `../../../../external` relative to the project dir
  - Also: Visual Studio macros like `$(ProjectDir)`, `$(SolutionDir)`
- By default, VS runs your program in the $(ProjectDir)
  - Reference „external“ accordingly in your code!
  - Check for your output images there
  - To change this: Properties -> Debugging -> Working Directory

Command Line Parsing

- Command-line call:
  `TestProgram.exe -x 4096 -o "outputfolder\terrain_height.raw"`
- Your program:

```
int _tmain(int argc, _TCHAR* argv[])
```

Similar to Java, the main() function receives the command line arguments

- argc contains the number of arguments
- argv[] is an array of pointers to _TCHAR!

- In this example:

```
argc: 5
argv[0]: "C:\git\GEDSS17\reichlf\solution\Debug\TestProgram.exe"
argv[1]: "-x"
argv[2]: "4096"
argv[3]: "-o"
argv[4]: "outputfolder\terrain_height.raw"
```

Command Line Parsing

- Visual Studio specific: `_tmain()` and `_TCHAR`
  - Strings can be ASCII or Unicode…
  - Two character types: `char` and `wchar_t`
  - _TCHAR is defined depending on project settings
  - Unicode (`wchar_t`) is default!

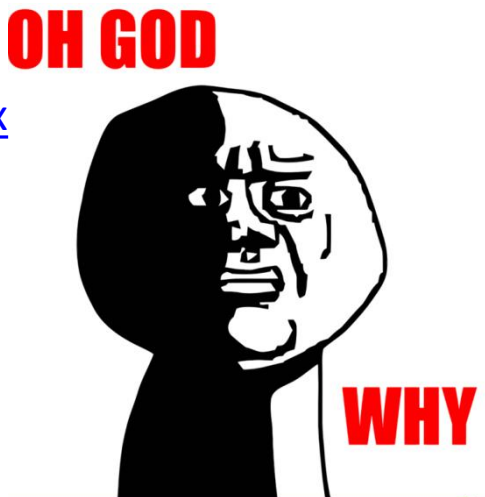
- For correct support
  - Use TCHAR instead of `char` / `wchar_t`
  - Enclose string literals you assign to a TCHAR* with TEXT(…)

```
TCHAR* cstring = TEXT("This is a C-String");
```

Command Line Parsing

- Remember: TCHAR* is just a simple pointer
  - You cannot compare C-Strings using ==
  - This will just compare the pointers!
  - Use _tcscmp()
    http://msdn.microsoft.com/de-de/library/vstudio/e0z9k731.aspx
    - Again: VC++ specific
    - Calls strcmp() or wcscmp() depending on charset
      http://www.cplusplus.com/reference/cstring/strcmp/
    - Returns 0 if the strings match!

- To convert TCHAR* to int: _tstoi()
  http://msdn.microsoft.com/de-de/library/yd5xkb5c(v=vs.80).aspx
  - VC++ specific…
  - Like atoi()
    http://www.cplusplus.com/reference/cstdlib/atoi/



Command Line Parsing

- Oh god…
  - … we‘ll just give you example code for this ;-)
  - This is why you use `std::string` whenever possible!

```
// ... remember to check the correct number of arguments before!

// Compare if the first argument after the executable name matches "-r"
if (_tcscmp(argv[1], TEXT("-r")) != 0)
{
	// Incorrect parameter... do something here!
}

int resolution = _tstoi(argv[2]);
```

Command Line Parsing

- If possible: std::string for char*, std::wstring for wchar_t*

```
std::wstring ws(argv[0]);                // wchar_t* to std::wstring
const wchar_t* cstr = ws.c_str();        // std::wstring to (const) wchar_t*
std::string s(ws.begin(), ws.end());     // std::wstring to std::string... and vice versa!
```

- To extract various data types from a string:
  `std::stringstream` / `std::wstringstream`

#include <sstream>

```
std::wstringstream wsstream;
wsstream << argv[2];

int resolution;
wsstream >> resolution;
```

Utilities

- Image handling utilities: GEDUtils::SimpleImage
  #include <SimpleImage.h>

  – Use this to save your generated heightfields, color- and normal maps
  – Copy your heightfield into a SimpleImage: setPixel(x, y)

  – See external/SimpleImageSample.cpp for an example

Utilities

- Colors and normals will be generated in the next assignment
  - Use `GEDUtils::TextureGenerator` for now
  - See header file for comments
  - Fast path:
    `GEDUtils::TextureGenerator::generateAndStoreImages()`

Error Checking

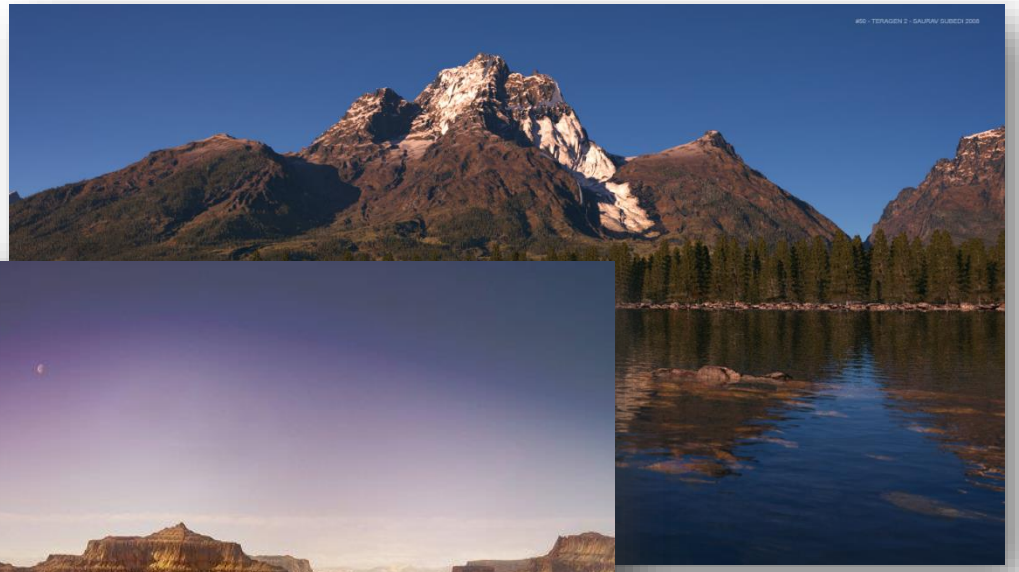
- Exceptions
  - Derived from std::exception
  - Get the error message by:

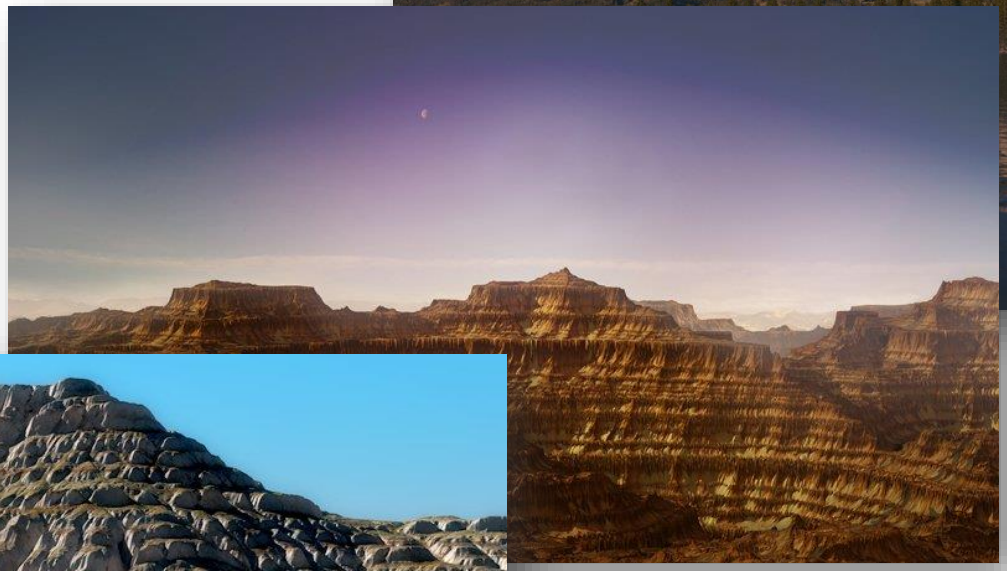
```
try
{
    heightImage.saveToFile(output_height_filename);
}
catch (std::exception& e)
{
    std::cout << e.what();
}
```

- Assertions
  - Debugging only!
  - `assert(cond)`: Debugger will stop if `cond == false`
  - To (sort of) include an error message:
    `assert(cond); //Something went horribly wrong`

Fractal Landscapes

Fractal Landscapes



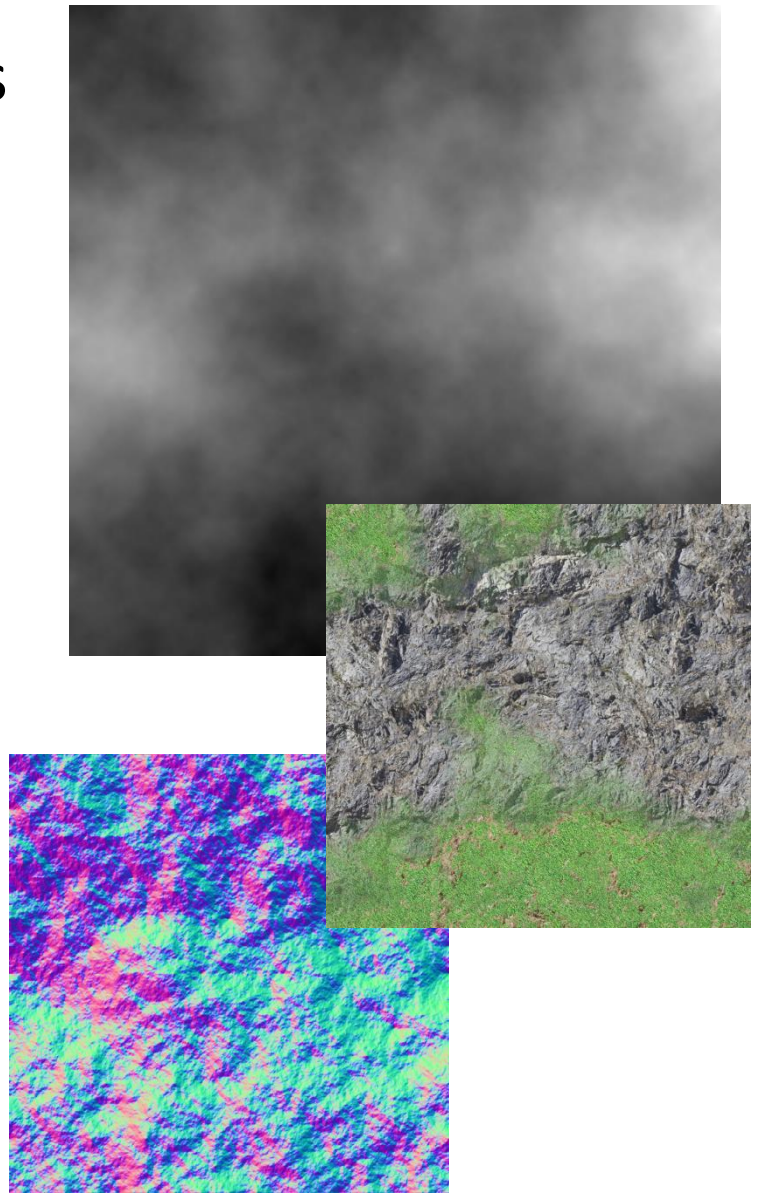


Commercial Tools

- Terragen http://www.planetside.co.uk
- WorldMachine www.world-machine.com
- L3DT (freeware) www.bundysoft.com
- GeoControl www.geocontrol2.com
- … (many more)

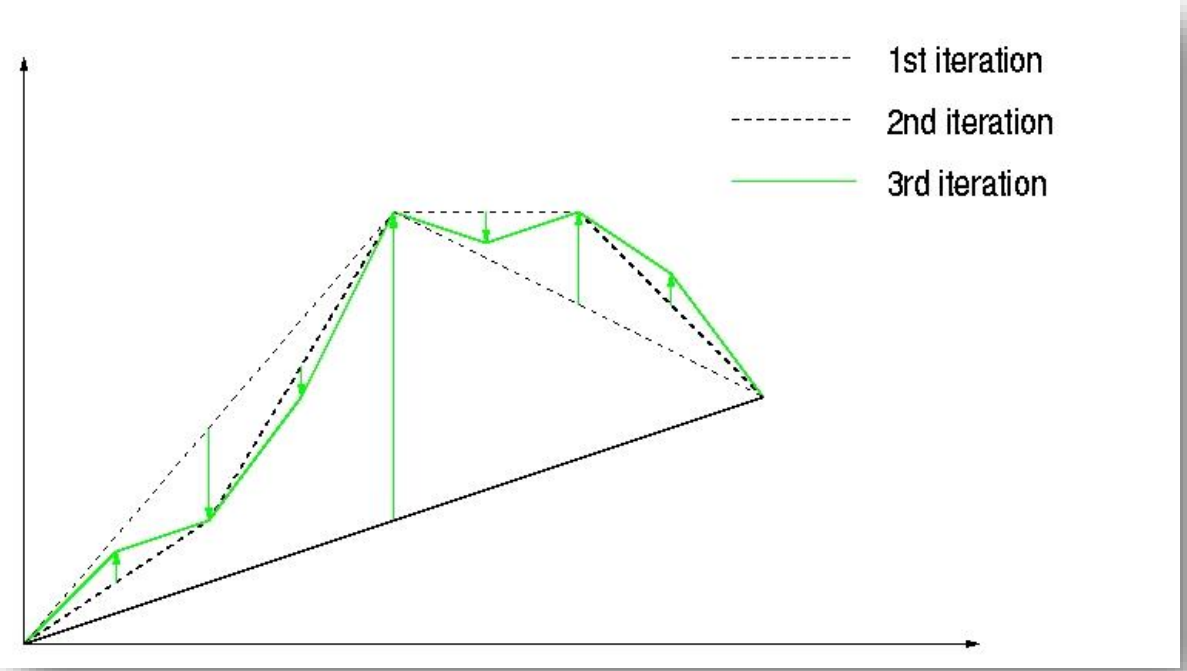
Terrain Generation

- Fractal landscapes are created as heightfields
  - Regular 2D grid contains „height above ground“
- Additional 2D maps needed
  - Texture map for realistic look
  - Normal map for more details
  - These will be created in the next assignment



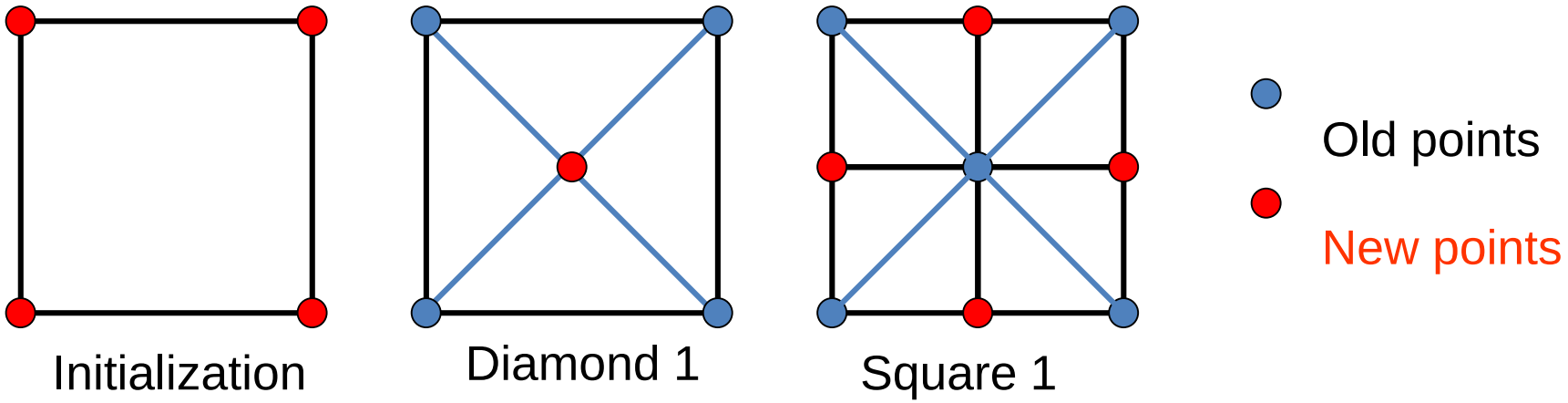
Midpoint Displacement

- Random Midpoint Displacement
  - Interpolation + random displacement
  - Displacement proportional to iteration count



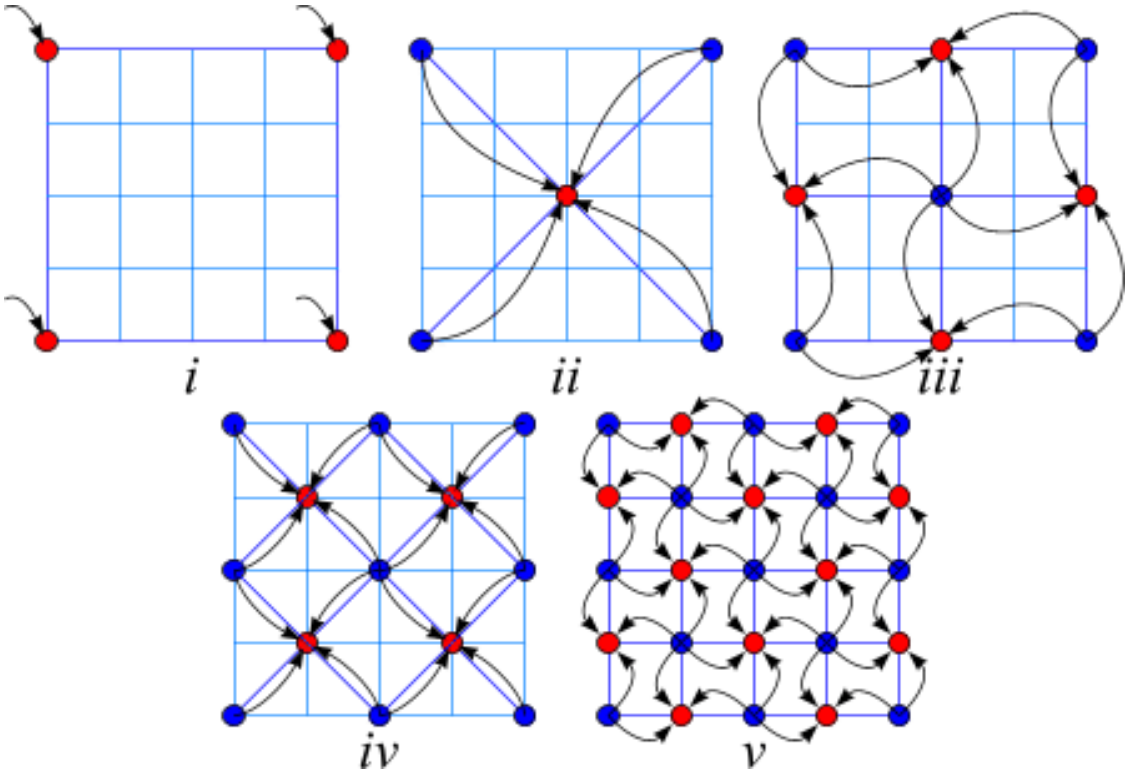
Midpoint Displacement

- Diamond-Square-Algorithm
  1. Assign (random) values to the corners
  2. (**Diamond**) Assign the average of four corners plus a random displacement in $(-R^i, R^i)$ to the midpoint
     - R: *Roughness* in (0, 1), i: *Current iteration*
  3. (**Square**) For each diamond, average four corners and add a random displacement in $(-R^i, R^i)$
  4. Repeat step 2 and 3 for a given number of iterations



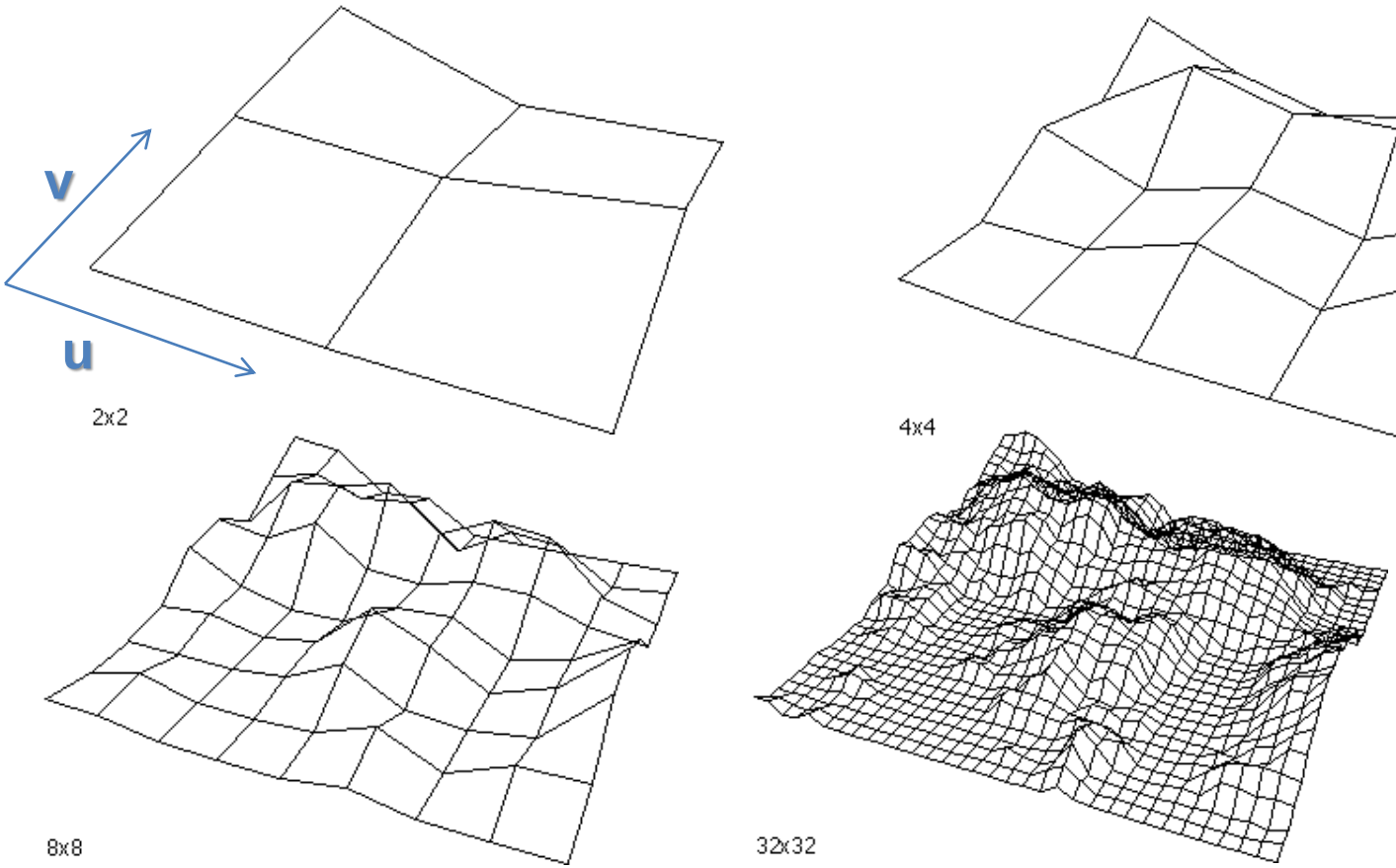
Midpoint Displacement

- Properties
  - Height depends on neighboring quads
  - Large height differences are result of early iterations



Midpoint Displacement

Step-by-step refinement in 2D:

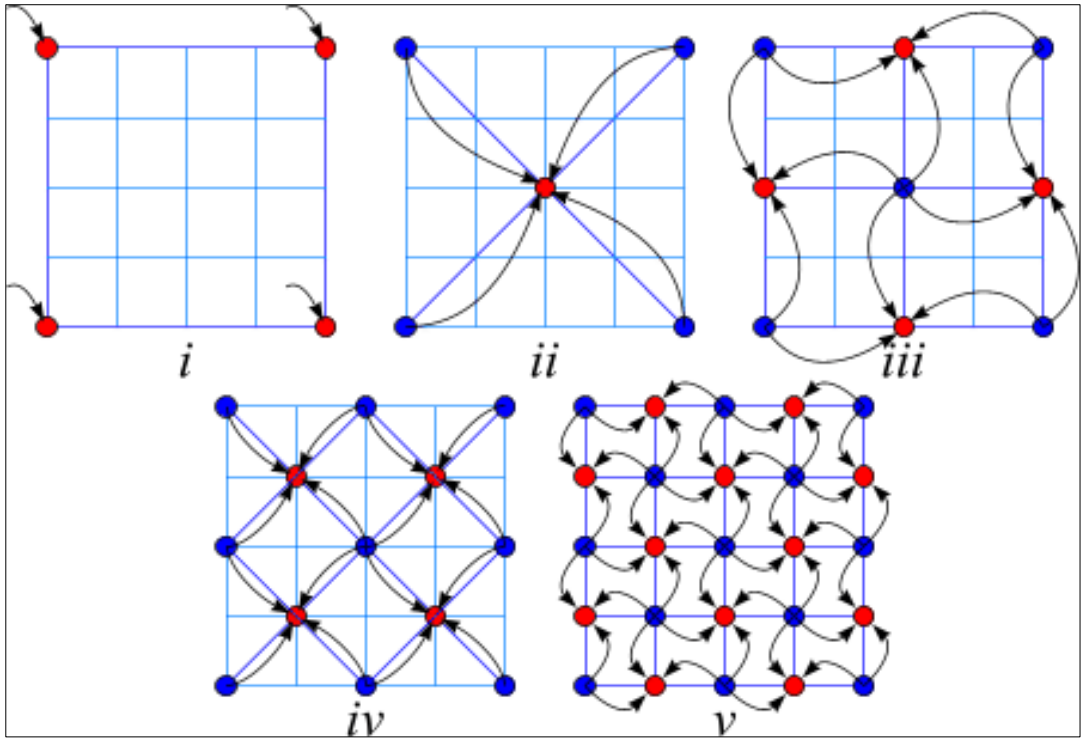


Diamond-Square-Algorithm

- Implementation idea:

```
Initialize(…); //i

int r = m_resolution;
for(int s=r/2; s>=1; s/=2) {
  for(int y=…) {
    for(int x=…) {
      //ii, iv, …
      Diamond(x,y,s,…);
    }
  }
  for(int y=…) {
    for(int x=…) {
      //iii, v, …
      Square(x,y,s,…);
    }
  }
}
```



Advanced Random Numbers

- C++11 introduces more advanced randomization functions
  - `#include <random>`
    http://www.cplusplus.com/reference/random/
- Generator: Stores the „state“ of random number generation
  - Create a single instance of std::default_random_engine()
  - As before, seed with time()
    - This is passed to the constructor
- Distribution: Generates random numbers of a specific distribution
  - Uses the random sequence created by the generator

Advanced Random Numbers

- We need to create normally distributed random numbers
  - std::normal_distribution<float>
    http://www.cplusplus.com/reference/random/normal_distribution/
  - Mean: Current expected midpoint
    - e.g. 0 to generate a value between –x and +x…
  - Sigma: Standard deviation
    - Will determine the „range“ of your numbers
    - Depends on the current iteration (R^i)
    - To generate only values in [-x, +x]: While result not in [-x, +x], generate a new number!

Coding Hints

- Try to seperate your algorithm into multiple methods
  - Seperate DiamondSquare method / class
  - Keep your methods small (up to 50 lines)
    - diamondStep()
    - squareStep()
    - random(min, max)
    - getHeight(x, y)
    - setHeight(x, y)
    - etc…
- Remember: Save a power-of-two heightfield (discard the last row / column after everything is done)

Refinement

- Some hints for nice looking terrain:
  - Try multiple smoothing passes
  - Try different smoothing radii (larger than 3x3)
  - Change the roughness parameter *R*
  - Change the standard deviation by a fixed factor *k* in *[0.0, 2.0]*
  - Most important: Try multiple combinations of all of the above!

References

- Diamond-Square-Algorithm:
  - http://en.wikipedia.org/wiki/Diamond-square_algorithm

Questions?

Remember that you can always ask questions in our Q&A forum or you can ask your tutor.

https://qage.in.tum.de/