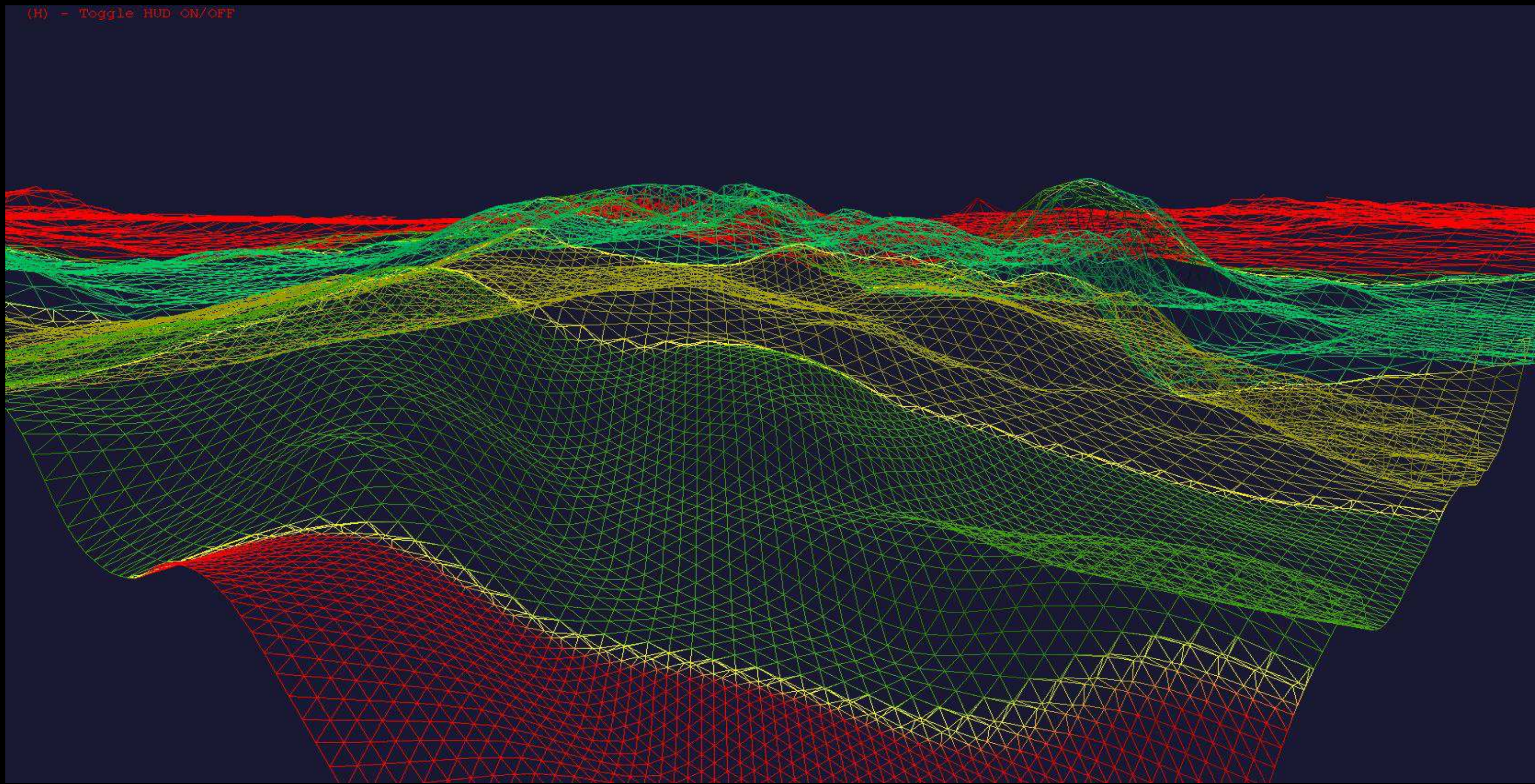
Procedural Terrain Generation using a Level of Detail System and Stereoscopic Visualization

Octavian Mihai Vasilovici
MSC Project
Bournemouth University 2013

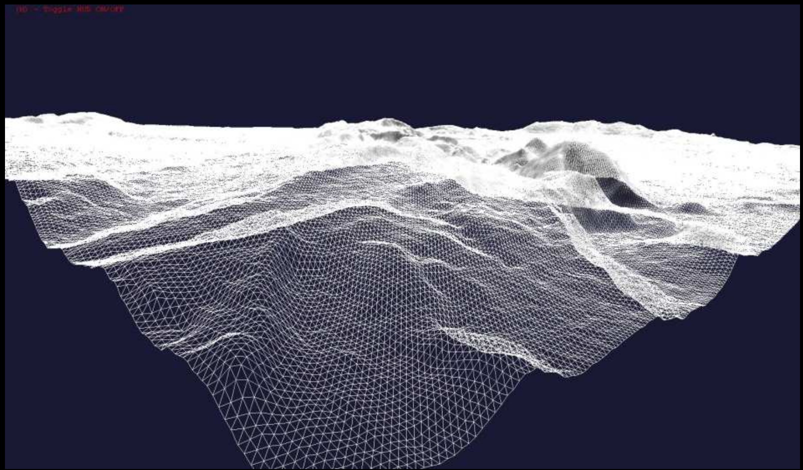


- Terrain using Continuous Distant-Dependent Level of Detail

Overview

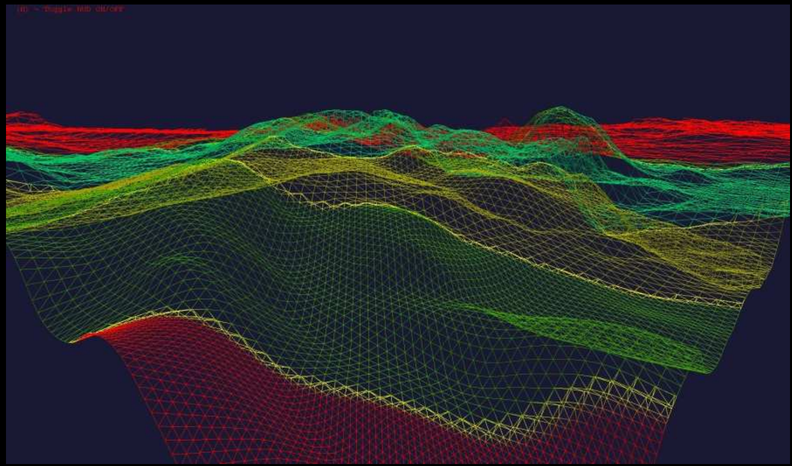
- Why is it needed ?
- Existing algorithms – a quick overview
- CDLOD – Strengths and Weaknesses
- CDLOD – Implementation
- Stereoscopy – a quick overview
- Stereoscopic pipeline
- A word about stereoscopic performance
- Instanced Cloud Reduction
- See it in action
- Conclusions
- Questions

Why is it needed ?



Performance:16 FPS
Grid: 1024x1024
No LOD
Triangles: 8.388.608

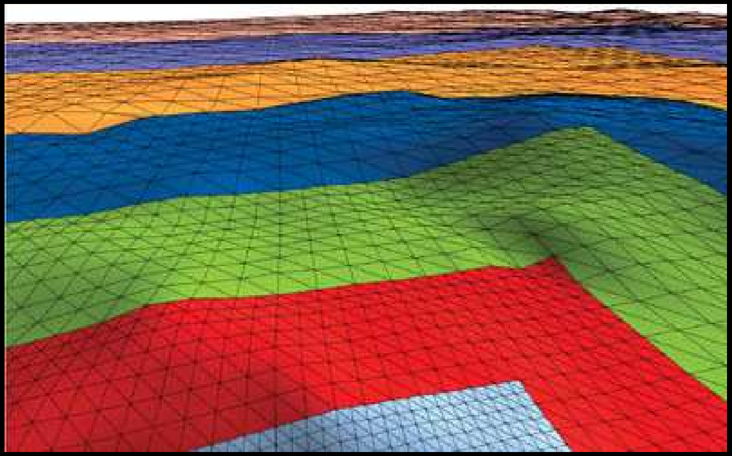
Unusable for real-time



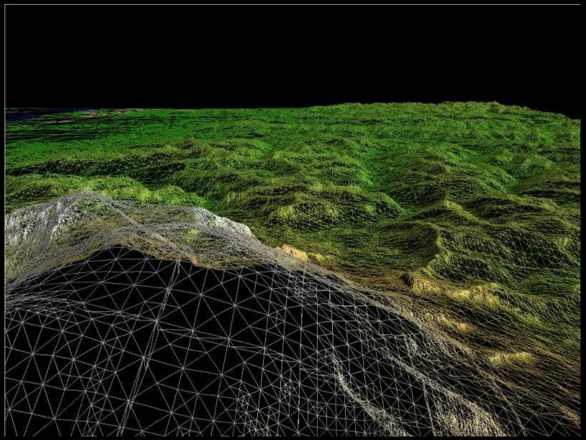
Performance:1200 FPS
Grid: 1024x1024
5 LOD Levels
Triangles: 56.576

More than perfect for real-time

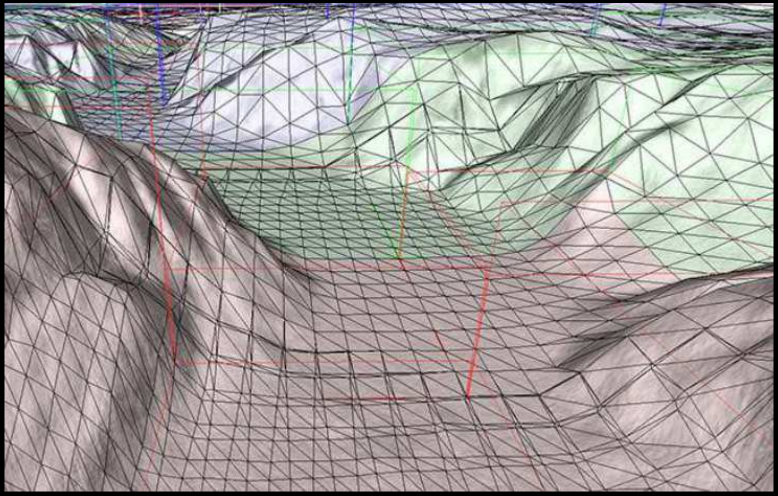
Existing algorithms – a quick overview



Terrain using ClipMaps – Asirvatham A. et al. 2005



Terrain using CLOD – Ulrich T., 2002



Terrain using CDLOD – Strugar P., 2010

CDLOD – Strengths and Weaknesses

Strengths:

- GPU-based
- Easy Implementation and integration with other methods
- Can easily be extended with a streaming mechanism such as ROAM
- LOD Selection based on viewer position in 3D space (unlike ClipMaps alg.)
- No need for additional geometry for stitching gaps (unlike CLOD alg.)
- Continuous Morphing in Vertex Shader
- Bounding boxes used for quadtree partitioning require only 2 values: min and max

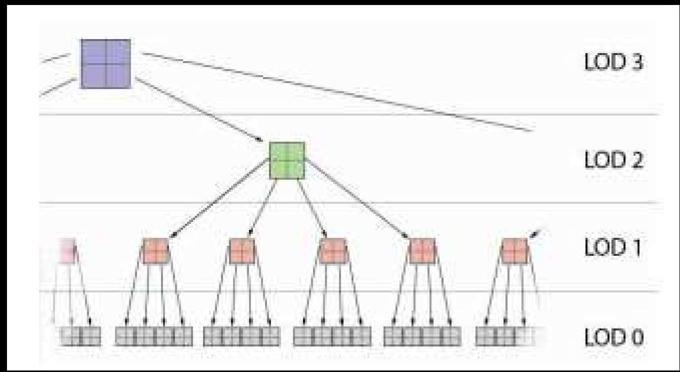
Weaknesses:

- Can only select between 2 LOD levels at once:
- – limits the maximum viewing range or minimum quadtree depth;
- – increasing the LOD brake (viewing range) can fix the cracks that appear on very uneven terrain, but at the cost of extra data to be rendered.
- – reducing the LOD levels can also fix this problem but makes the algorithm less effective
- Recursive algorithm* for LOD selection
- Still heavy memory bound
- Requires extra tuning for each dataset in order for the algorithm to be effective

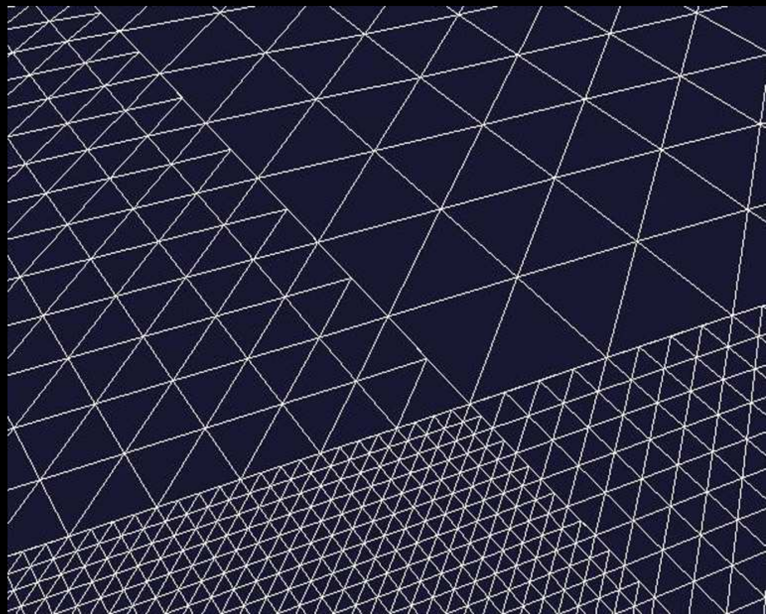
CDLOD – Implementation

- ❖ Implementation done in OpenGL, QT, NGL based on translation of DirectX source code.
- ❖ Terrain generation as a pre-step and each LOD level saved in the quadtree
- ❖ Terrain rendered as a series of triangle strips
- ❖ Support for 8 LOD levels
- ❖ Support for any HeightMap texture (both greyscale or color)
- ❖ Grid size, grid scale, node size, texture map, LOD view distance parameters fully exposed to the user via the Setup Wizard.
- ❖ Ability to Load and Save presets.
- ❖ Currently supports only square grids
- ❖ Heightmap texture needs to have the same resolution as the grid for correct sampling

CDLOD – Implementation



Quadtree and LOD levels – Strugar F., 2010



LOD levels as seen in the application. Three additional LOD levels

CDLOD – Implementation

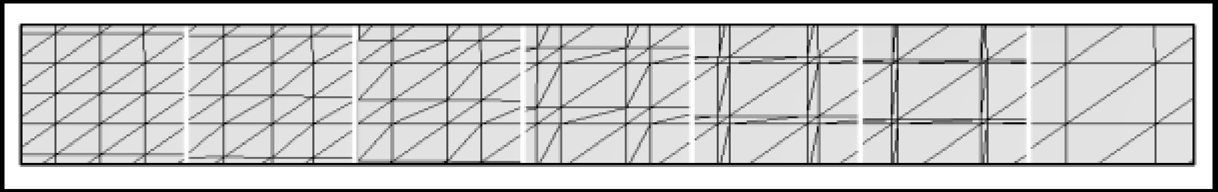
- Morphing:
  - Values between 0 and 1 (no morph, full morph)
  - Done in the vertex shader (for each vertex)
  - A morph vertex is defined as a grid vertex having both indexes an odd number (Strugar F., 2010)
  - A no-morph vertex is defined as a grid vertex with indexes (i-i/2, j-j/2) (Strugar F., 2010)

- Implementation in GLSL:

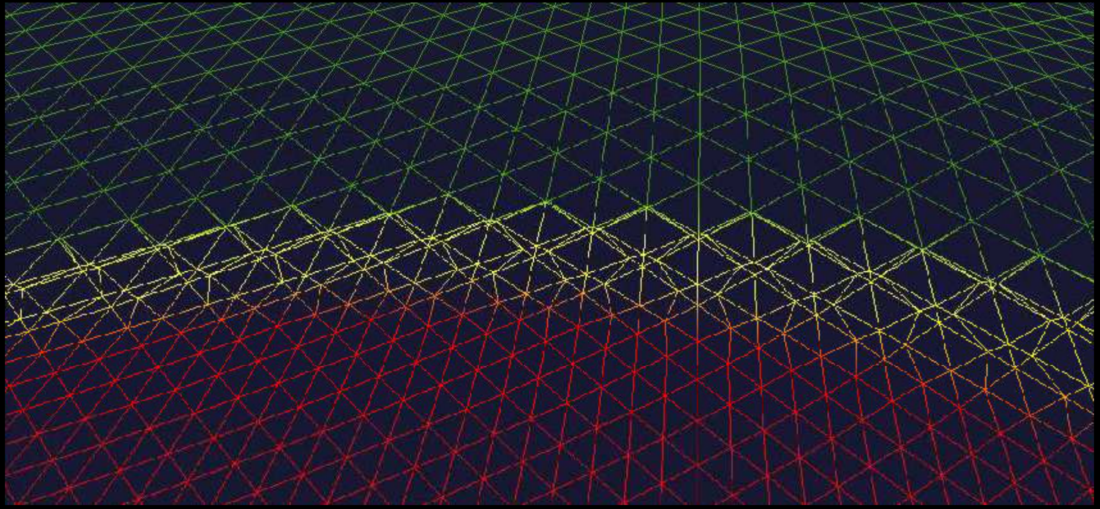
```
// Calculate the vertex position based on the VertexID and InstanceID
   int thisX = gl_VertexID % stride + baseX;
   int thisY = gl_InstanceID + gl_VertexID / stride + baseY;
   vec2 thisXY = vec2 (float (thisX), float (thisY)) * texScale * lodStep;
// Calculate the morphing
   if (thisDist > mixStart)
   {
    //bitwise operator (and not)
    int evenX = thisX & ~1;
   int evenY = thisY & ~1;
   mixFactor = clamp ((thisDist - mixStart) / mixWidth, 0, 1);
    if ((evenX != thisX) || (evenY != thisY))
      {
         vec2 evenXY = vec2 (float (evenX), float (evenY)) * texScale * lodStep;

         thisXY = mix (thisXY, evenXY, mixFactor);
         thisZ = texture (heightTexture, thisXY + texBias).r;
         thisPosition = vec3 (mv * vec4 (vec3 (thisXY, thisZ) * scale, 1));
      }
```

CDLOD – Implementation



Continuous Morphing – Strugar F., 2010

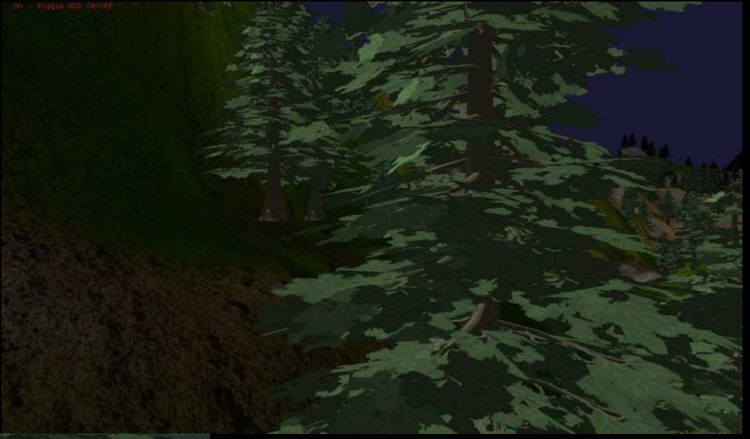


Continuous Morphing as seen in the application

Stereoscopy – a quick overview

- Scene is rendered as Left and Right view with a slight offset between

Left Eye

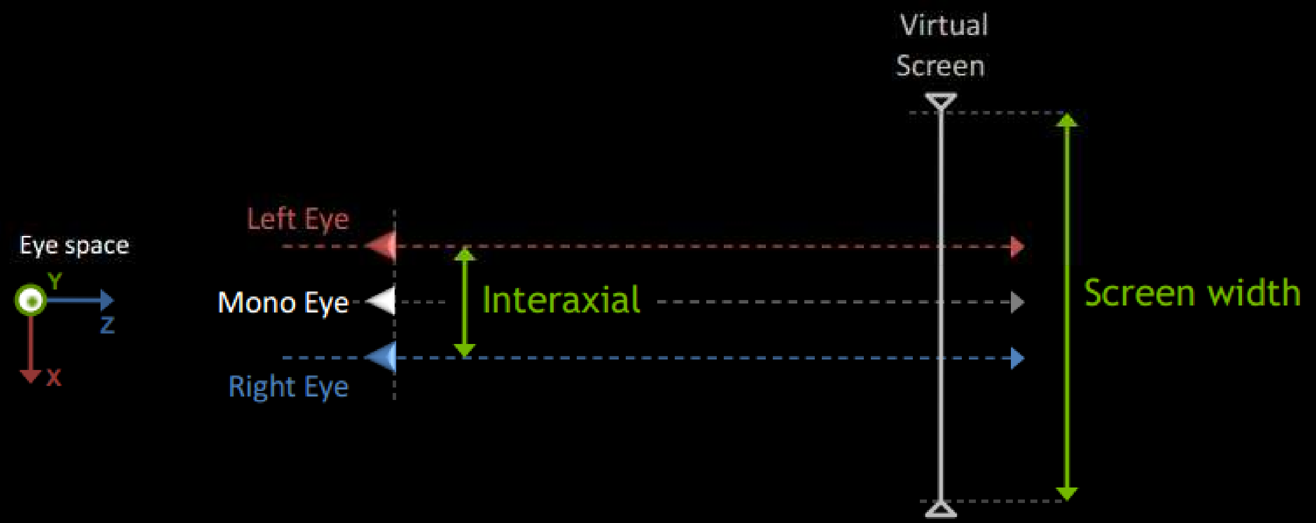


Right Eye

Both Eyes

Stereoscopy – a quick overview

- Key concept: Depth
  - Separation controls the how the depth of objects is perceived

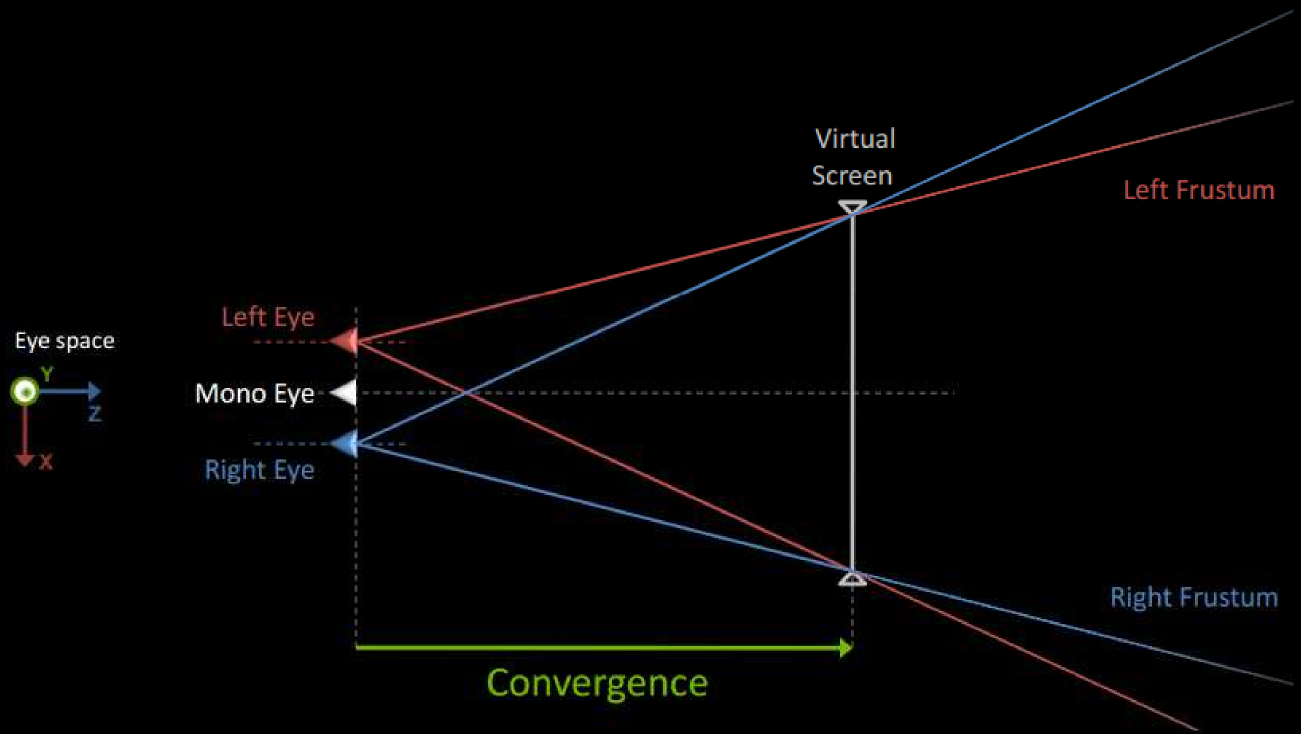


Separation = Interaxial / Screen Width

Implementing Stereoscopic 3D in Your Applications. nVidia Corporation. 2010

Stereoscopy – a quick overview

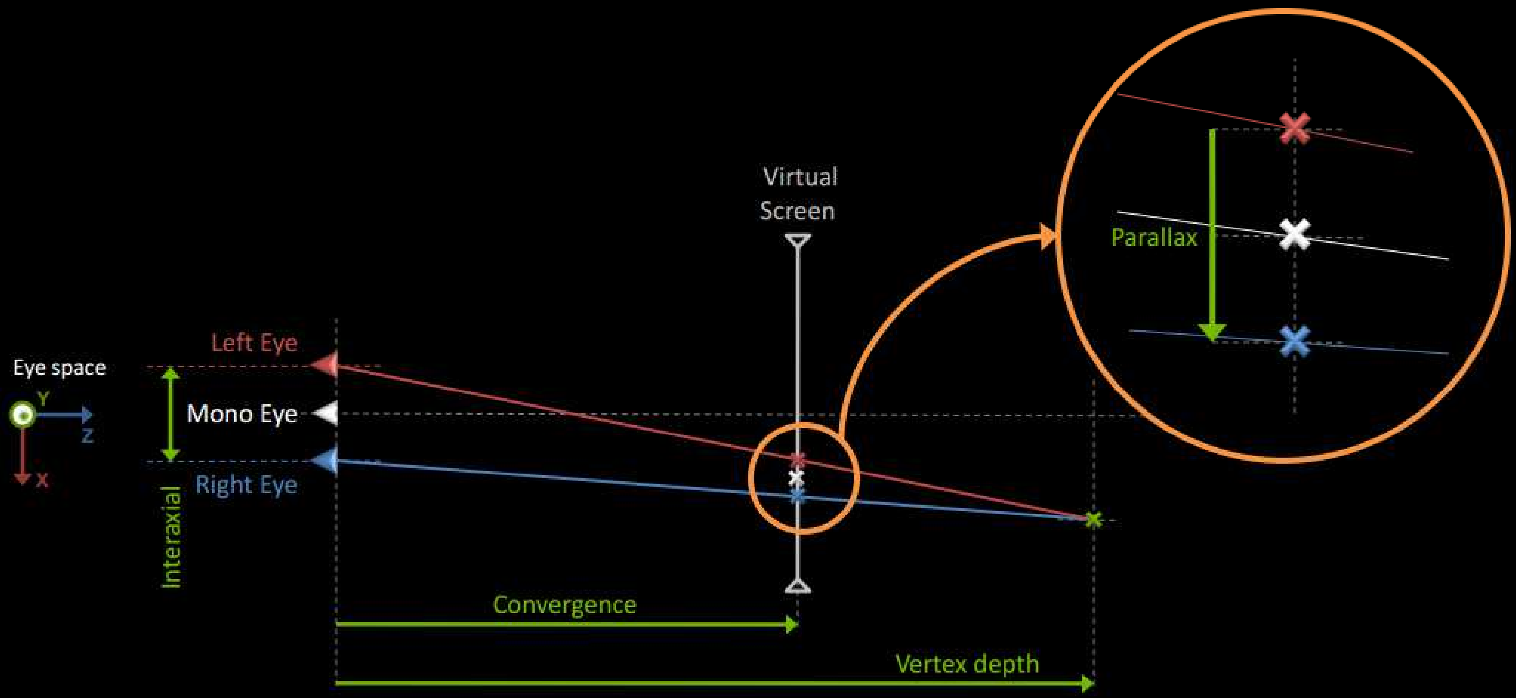
- Key concept: Convergence
  - Convergence controls the distance to zero Parallax Plane



Implementing Stereoscopic 3D in Your Applications. nVidia Corporation. 2010

Stereoscopy – a quick overview

- Key concept: Parallax
  - Parallax is responsible for "Push-in" and "Pop-out" effects of objects:
    - Objects closer than the zero Parallax plane will have a "Pop-out" effect from the screen
    - Objects at zero Parallax will be at "screen level"
    - Objects farther than the zero Parallax plane will have a "Push-in" effect in the screen



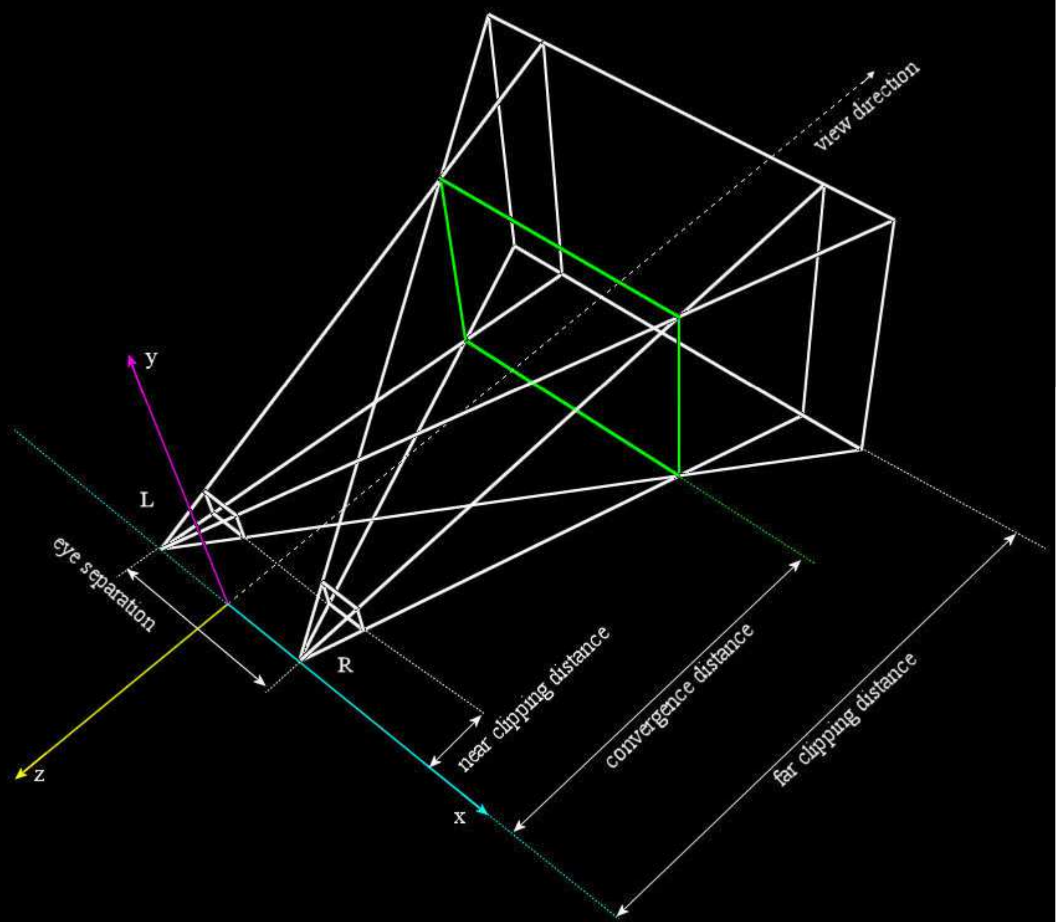
Implementing Stereoscopic 3D in Your Applications. nVidia Corporation. 2010

Stereoscopic pipeline

- Requires creation of a stereoscopic context in the application
- Requires two virtual cameras
- Requires two Projection matrices with an offset
- Each camera has an offset equal to +-(Separation /2) from the monoscopic camera location

Implementation done by expanding the ngl::Camera() class

- Camera transformations are done on the monoscopic camera to avoid problems like eye inversion during rotations
- Camera positions & Projections are updated per frame
- Rendering is done using QuadBuffering technique



Mishra A., 2011.

A word about stereoscopic performance

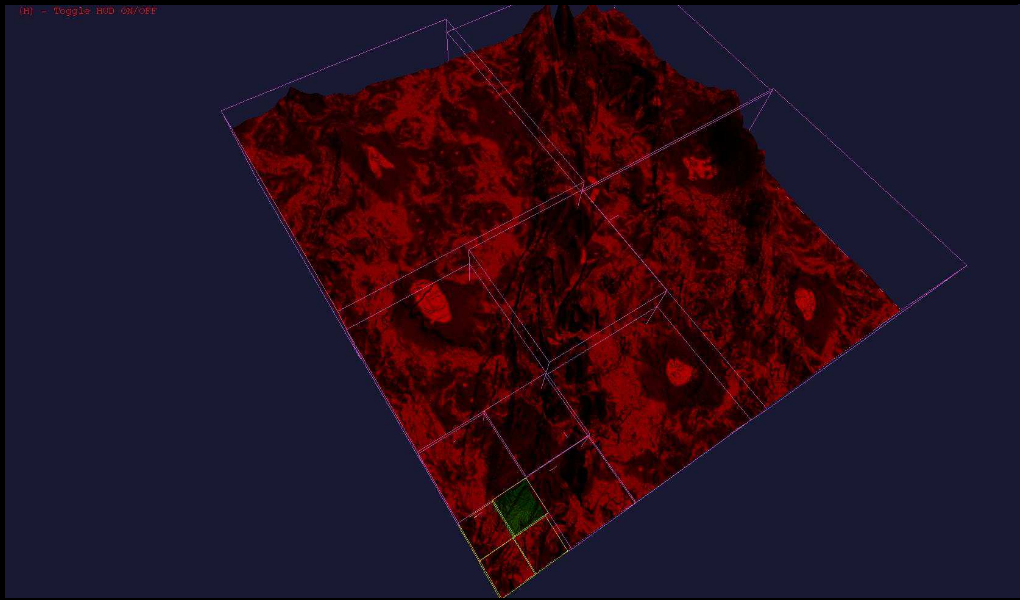
- The scene must be rendered two times (with some exceptions):
  - For left eye
  - For right eye
- Performance is theoretically cut by 50%
- In practice this is not always the case
- The performance cut can be remedied by using a dual-GPU configuration
- Strongly dependent on hardware limitations and driver implementation
- Behavior is different in forward and deferred rendering
- Does not employ artificial techniques like using the Depth Buffer for object depth calculation

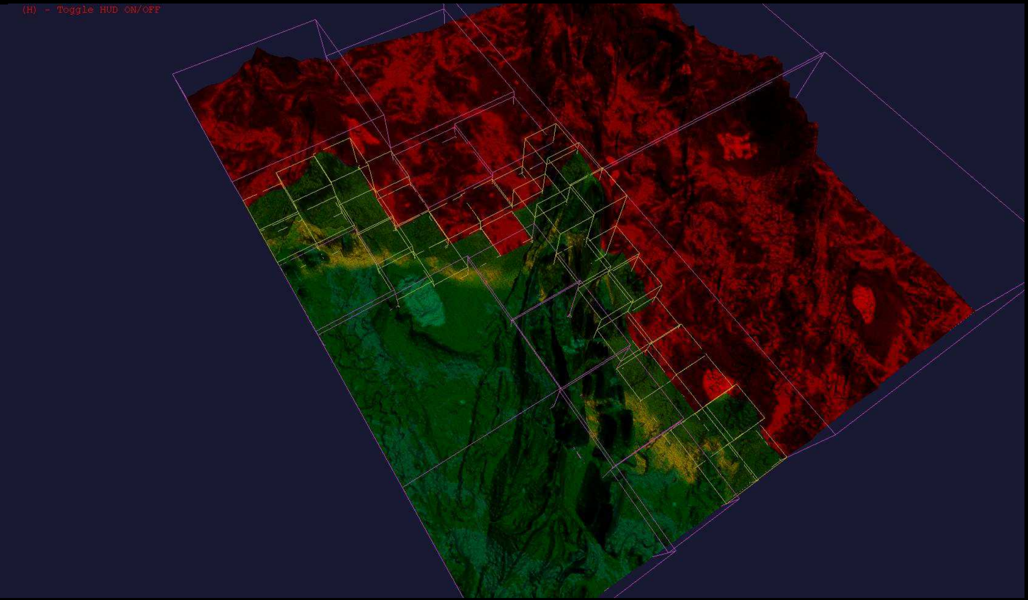
Instanced Cloud Reduction

- Algorithm employed to simulate a real case scenario.
- Instanced Rendering
- Culling done in geometry shader
- Position of object done randomly (X, Z) and based on the HeightMap information (Y)
- In combination with stereoscopy greatly reduces the framerate, close to a real case scenario

See it in action

Screen Shots

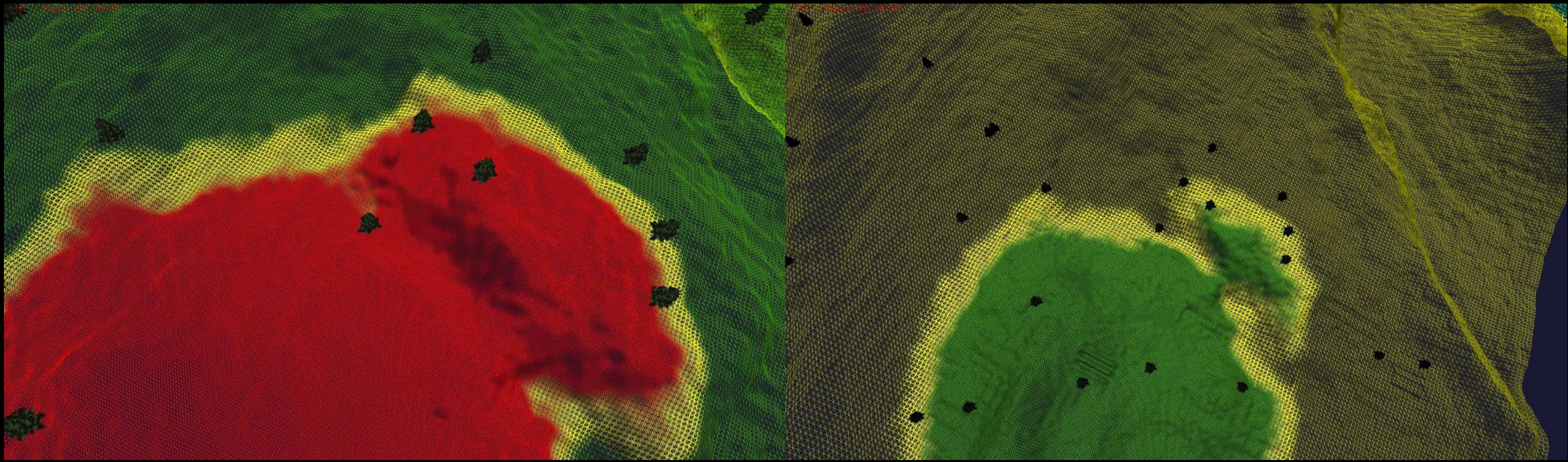




Quadtree Traversal and LOD Selection

See it in action

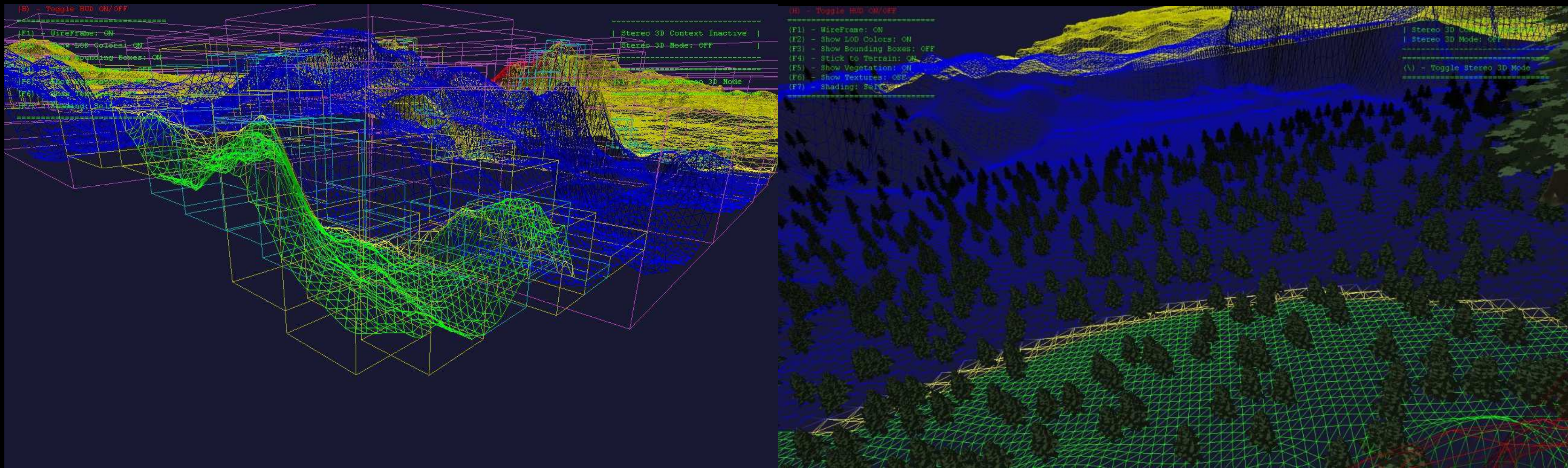
Screen Shots



LOD Selection based on Height

See it in action

Screen Shots



LOD View

See it in action

Screen Shots

Screenshots as seen in the application

See it in action

Stereoscopic Rendering Live Demo

Conclusions

- Advantages:
  - Possible to render real-time detailed terrains when using a LOD system
  - Huge Performance gain
  - Predictable LOD system, with no need for additional geometry to fix seams or gaps due to the continuous morphing system
  - Stereoscopy greatly improves the visualization quality

- Disadvantages:
  - Algorithm must be tweaked for each dataset
  - Highly dependent on memory allocation space
  - Stereoscopy theoretically cuts the performance by 50%
  - Hardware and driver bound

References

- Asirvatham A., and Hoppe H., 2005. *GPU Gems 2. Addison-Wesley Professional*
- Bourke P., 1999. *3D Stereo Rendering Using OpenGL (and GLUT). Available from:* ftp://ftp.sgi.com/opengl/contrib/kschwarz/GLUT_INTRO/SOURCE/PBOURKE/index.html [Last accessed August 2013]
- Cervin A., 2012. *Adaptive Hardware-accelerated Terrain Tessellation. Available from:* http://dice.se/wp-content/uploads/adaptive_terrain_tessellation.pdf [Last accessed August 2013]
- Gateau S. and Nash S., 2010. *Implementing Stereoscopic 3D in Your Applications. Nvidia Corportation. Available* from: http://www.nvidia.com/content/GTC2010/pdfs/2010_GTC2010.pdf [Last accessed August 2013]
- GeForce Forums., 2013. *[Test Request]Stereo 3D OpenGL application. Available from:*
- https://forums.geforce.com/default/topic/572432/3d-vision/-test-request-stereo-3d-opengl-application/ [Last accessed August 2013]
- Macey J., 2012. *Operator overloading and Dynamic Data Structures. Available from:* http://nccastaff.bournemouth.ac.uk/jmacey/ASD/slides/Lecture4OperatorsandDynamicDS.pdf [Last accessed August 2013]
- Mishra A., 2011. *Rendering 3D Anaglyph in OpenGL. Available from:* http://quiescentspark.blogspot.co.uk/2011/05/rendering-3d-anaglyph-in-opengl.html [Last accessed August 2013]

References

- Rakos D., 2010. *Instance culling using geometry shaders. Available from:* http://rastergrid.com/blog/2010/02/instance-culling-using-geometry-shaders/ [Last accessed August 2013]
- Strugar F., 2010. *Continuous Distance-Dependent Level of Detail for Rendering Heightmaps (CDLOD). Available* from: http://www.vertexasylum.com/downloads/cdlod/cdlod_latest.pdf [Last accessed August 2013]
- Ulrich T., 2002. *Rendering Massive Terrains using Chunked Level of Detail Control. Available from:* http://tulrich.com/geekstuff/sig-notes.pdf [Last accessed August 2013]
- Wloka M., 2004. *Optimizing the Graphics Pipeline. Nvidia Corporation. Available from:* https://developer.nvidia.com/sites/default/files/akamai/gamedev/docs/EG_04_OptimizingGPUPipeline.pdf [Last accessed August 2013]
- Wolfgang E., and Mistal B., 2013. *GPU Terrain Subdivision and Tessellation.GPU Pro 4. A K Peters/ CRC Press*

Questions

Any questions ?

THANK YOU !