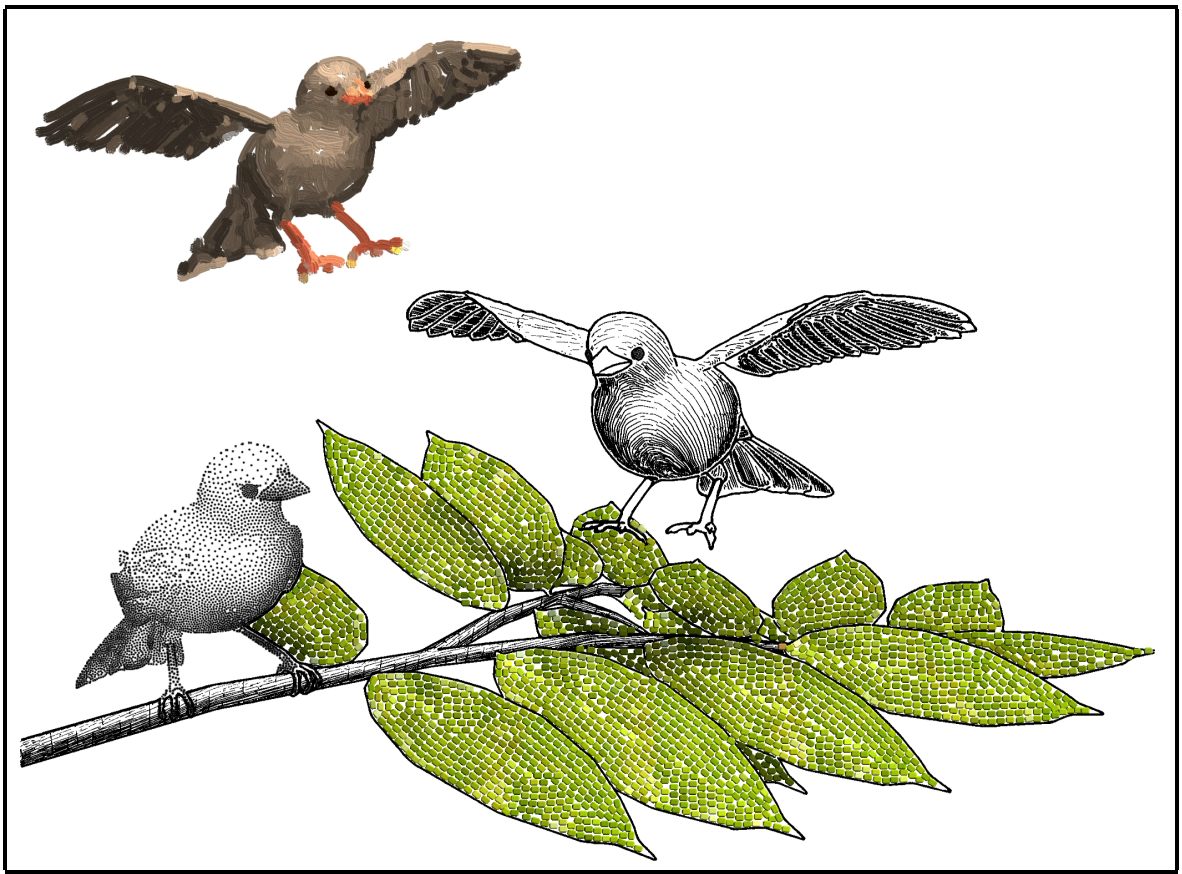
NORSIGD INFO

Nummer 1 2005



NORSK SAMARBEID INNEN GRAFISK DATABEHANDLING

ISSN 0803-8317

Aktivitetskalender

Hva skjer når og hvor?

Juni 2005

20–24 **GraphiCon 2005**: Fifteenth International Conference on Computer Graphics and Applications, Novosibirsk Akademgorodok, Russland. `http://www.graphicon.ru/2005/`.

29–1 **Eurographics Symposium on Rendering 2005**: 16th Eurographics Workshop on Rendering, Konstanz, Tyskland. `http://www.cgmi.inf.uni-konstanz.de/egsr2005/`.

Juli 2005

2–6 **Eurographics/ACM SIGGRAPH 3rd Symposium on Geometry Processing**: , Wien, Østerrike. `http://www.geometryprocessing.org/`.

7–8 **VISION, VIDEO AND GRAPHICS 2005**: , Edinburgh, UK. `http://www.ima.org.uk/mathematics/vvg.htm`.

31–(4) **SIGGRAPH 2005**: 32st Int'l Conf. on Computer Graphics and Interactive Techniques, Los Angeles, CA, USA. `http://www.siggraph.org/s2005/`.

August 2005

28–29 **SBM 2005**: 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling, Trinity College, Dublin, Irland. `http://www.eg.org/sbm/`.

29–(2) **EG 2005**: 27th annual conf. of the European Association for Computer Graphics (EUROGRAPHICS), Trinity College, Dublin, Irland. `http://isg.cs.tcd.ie/eg2005/`.

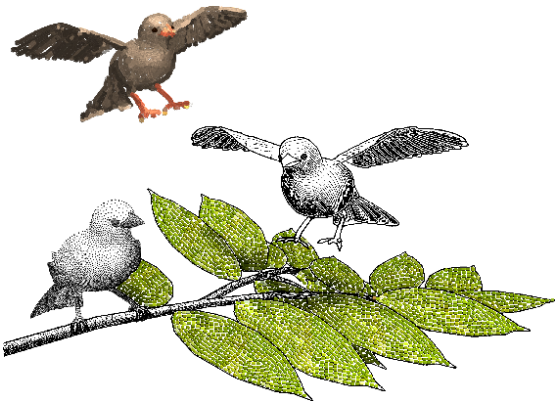
October 2005

23–28 **VIS2005**: IEEE Visualization 2005, Minneapolis, MN, USA. `http://vis.computer.org/vis2005`.

Web-Kalender

Helwig's Conference Calender: `http://www.vrvis.at/ConfCal/`

Eurographics Events: `http://www.eg.org/events`



Om forsiden
Treebirds: Bildet viser et eksempel for et bilde laget med RenderBots. Disse er beskrevet i artikkelen på side 4.

Hilsen fra styret

Kjære medlemmer,

Det finnes mange spennende applikasjoner innen grafisk databehandling som er nyttige innen ulike fagfelt. Datagrafikk kan brukes til bl.a. illustrasjoner, leting etter olje, medisin og simuleringer. I denne utgaven presenterer vi igjen godbiter fra vårt fagfelt.

I første artikkel presenteres Renderbots som en ny metode innen NPR. Så får vi tredje og siste del av artikkelen om medisinske anvendelser innen virtuell endoskopi. Deretter gis et innblikk i hvordan turbiditter modelleres. Disse brukes innen geologi bl.a. for leting etter olje.

Bruk av GPUer og nye grafikk-kort for andre formål enn spill, er et spennende tema som bearbeides hos SINTEF avd. for Industriell Matematikk for tiden. Her presenteres grunnprinsippene for GPU-programmering og hvilke teknikker som brukes når GPUer brukes til implementering av ulike simuleringer, bl.a. av bølger som forårsakes av naturkatastrofer.

Undertegnede skriver om sitt besøk på konferansen SIMVIS 2005 i Magdeburg. I tillegg presenteres løsninger fra kurset “Grafiske Metoder” ved Høyskolen i Bergen. Det er nå andre gang vi får innblikk i datagrafikk-undervisningen på Høgskolen i Bergen.

Hilsen,

Wolfgang Leister

NORSIGD Info
– medlemsblad for NORSIGD

Utgitt av:	NORSIGD
Ansvarlig:	Wolfgang Leister Norsk Regnesentral Postboks 114 Blindern 0314 OSLO
ISSN:	0803-8317
Utgivelser:	2005: 15/5 15/11
Annonsepriser:	Helside kr 5 000 Halvside kr 2 500
Layout:	Wolfgang Leister LaTeX2ε

Innhold

RenderBots—Multi Agent Systems for NPR

Tobias Germer, Stefan Schlechtweg,
Department of Simulation and Graphics, Otto–von–Guericke University of Magdeburg

This article presents a new approach for stroke-based non-photorealistic rendering. RenderBots are individual agents each of which in general represents one stroke. They form a multi agent system and undergo a simulation to distribute themselves in the environment. Each RenderBot is able to execute a painting function which determines the appearance of the final image.

The development of non-photorealistic rendering has brought up a wealth of new image generation methods. Many of these techniques take a source image and re-render it using picture elements that are larger than a pixel (strokes). These so called stroke-based rendering algorithms [1] are well suited to create images that embody qualities of hand-made images, be this their look or their use for communication purposes. Our goal is it to present a unified framework for stroke-based rendering on the basis of image data that is capable of:

- creating various styles like stippling, hatching, painterly rendering, and mosaics,
- mixing these styles in one image,
- easily deriving new styles.

System Overview

The basic idea which we are following is the use of multi agent systems. A Multi Agent System is employed to render the strokes where each agent in principle represents one stroke. Different styles are created by different classes of agents which can coexist in the environment.

The environment is represented by a source image together with G-buffer data (where available: intensity, edges, edge distances, depth, normal vector, texture coordinates, etc.). The agents access the environment in a pixel-based to get information about the image to be re-rendered. Even without additional G-buffers the algorithms produce satisfying results.

The behavior control for the autonomous agents is based on the "boids" model by Reynolds [2], i.e., mechanisms as separation, alignment, and cohesion are available. Each agent type implements its own additional strategies and – most important – a way to render a graphical representation of the strokes. Agents communicate with each other via sensing other agents' positions and via leaving marks in the environment. The user decides when the result is finished and, therefore, the simulation should be stopped.

RenderBot Classes

StippleBots: Each agent represents one stipple. An adaptive separation behavior with a separation distance proportional to the source intensity yields an equal stipple distribution. Cloning and dying enables an appropriate number of stipples also depending on target intensity.

EdgeBots: Each agent moves towards the closest edge in the source image (read from an edge buffer). If an edge is found, the agent starts following this edge leaving behind a visible trace. Additional optimizations make this procedure more effective.

HatchingBots: All agents move along a given main direction (either globally or given, for example, by texture coordinates). While moving they try to keep a minimal distance to neighboring agents. If an agent crosses a dark region in the source image, it leaves behind a visible trace. These traces are only drawn if there are not yet enough traces from other agents within a certain distance.

MosaicBots: Each agent represents a mosaic tile, a rectangular area that is filled with the medium color of the underlying pixels in the source image. Cloning and separation behavior fill the image with these rectangles while avoiding overlapping tiles. Tiles orient themselves following nearby edges. A 3D-effect makes the image more realistic.

PaintBots: Each agent first tries to find an empty area in the result image and starts drawing a stroke using the current source image color. It then moves on in the direction with the smallest color difference between its own color and the source image color. Strokes are rendered as textured quadrilaterals while an additional bump map yields realistic effects.

Summary

Our approach enables to "re-render" any source image in a stroke-based style. In the setup step the user creates a number of RenderBot swarms and selects appropriate parameters. RenderBots may be constrained to single objects or move freely over the whole image. If additional 3D information is

available in form of G-buffers, they can be used and yield advanced setup possibilities. During the simulation each RenderBot creates a specific output (strokes). The user decides when the simulation is to be stopped. A large parameter set determines the appearance of the result. Since all RenderBots share common behaviors, new styles can be easily realized. The use of the RenderBots framework enables a unified treatment of all known and possibly unknown stroke-based rendering styles and allows mixed styles in one image without additional compositing operations. For a more detailes description see [3].

References

[1] Aaron Hertzmann. A Survey of Stroke-Based Rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, July/August 2003.

[2] Craig W. Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics (Proceedings of ACM SIGGRAPH 83)*, 21(4):25–34, 1987.

[3] Stefan Schlechtweg, Tobias Germer, and Thomas Strothotte. RenderBots—Multi Agent Systems for Direct Image Generation. *Computer Graphics Forum*, 24(1), 2005. to appear.

Inntrykk fra konferansen SIMVIS 2005

Wolfgang Leister, Norsk Regnesentral

Undertegnede besøkte konferansen SIMVIS 2005 i Magdeburg som fant sted 3./4. mars 2005. Konferansen hadde to hovedtemaer: simulasjon og visualisering. Konferansen fant sted for 16-ende gang og har en tradisjon innen fagmiljøet.

Konferansen var delt i to parallelle sesjoner med papers, og to foredrag i plenum. Plenumsforedraget første dag ble holdt av Prof. Thomas Ertl fra Universität Stuttgart og handlet om nye trender innen visualisering, der også bruken av GPGU sto i forgrunnen. Plenumsforedraget andre dagen av Sven Spieckermann handlet om erfaringer fra en bedrift som tilbyr tjenester rundt simulasjonsteknologi til industrien.

Undertegnede holdt et innlegg om simulasjon av HikerNet, som er et peer-to-peer-nettverk for meldingsformidling som bygger på ad-hoc forbindelser mellom mobiltelefoner. Siden dette ble gitt i simulasjons-linjen fikk jeg ikke ansledning til å følge med sesjonen om visualisering innen medisinske applikasjoner.

I et av foredragene som jeg husker best ble det brukt "augmented reality" for design av biler. Her ble ekte biler ombygd slik at takkonstruksjonen er fjernet, mens sjåføren sitter med en VR-hjelm mens han styrer bilen. Dashbord, tak og utsyn mot gaten blir gjort synlig for sjåføren gjennom display i hjelmen. Dette blir gjort for å undersøke hvilke innvirkninger ulike konstruksjoner av tak og dashbord har.

I et annet foredrag innen "augmented reality" ble lagerarbeidere utstyrt med VR-hjelmer som brukes istedenfor manuelle avkrysningslister. Slik kan arbeiderne se et bilde av varer som plukkes for å unngå feilleveranser.

I en annen presentasjon ble det foreslått et system for automatisk illustrasjon av dokumenter. I en tekstbehandler kan det fremheves noen søkeord, og det kommer opp passende illustrasjoner som kan brukes i dokumentet automatisk. Illustrasjonene kan være tegninger gjennom en søkemotor på Internett, en database, eller 3D objekter som blir generert som passende illustrasjon.

Best paper award ble gitt til en forskningsgruppe fra Østerrike (Helmut Doleisch, Michael Mayer, Martin Gasser, Peter Priesching, Helwig Hauser) til en presentasjon av "Interactive Feature Specification for Simulation Data on Time-Varying Grids", som viser nye veier for visualisering av tekniske simuleringsresultater.

Artiklene som ble sendt inn til konferansen er gjengitt i en bok: Thamas Schulze, Graham Horton, Bernhard Preim, Stefan Schlechtweg (editors): "Simulation und Visualisierung 2005", SCS Publishing House, Erlangen, 2005, ISBN 3-936150-40-0. Mer informasjon finnes på `http://www.simvis.org`.

Konferansen var nokså oversiktlig med ca. 70 deltagere, de fleste fra tyskspråklige land og fra Sverige. Samtidig med SIMVIS 2005 var det flere workshops innen simulasjons-området.

Applications of Virtual Endoscopy - Part III

Dirk Bartz, WSI/GRIS - VCM, University of Tübingen

Minimally invasive procedures are of increasing importance in medicine because they have less deleterious effects on the patient. In particular, these procedures are used in gastroenterology, surgery, neurosurgery, (interventional) radiology, and many other fields.

In the past issues, I introduced into applications of virtual endoscopy, in particular of virtual colonoscopy, virtual bronchoscopy, and virtual ventriculoscopy. In this issue, I will complete the presentation on virtual endoscopy with virtual angioscopy.

Virtual Angioscopy

The blood circulation system is of special interest for physicians, since many injuries and diseases of this organ system can result in serious, potentially life-threatening conditions. Many of the diseases cause stenosis or aneurysms of the blood vessels. Both developments can be assessed using virtual endoscopy methods. These methods can be particularly useful in diagnostic applications where interior explorations using "real" endoscopic tools are not possible.

Angioscopy of Cerebral Blood Vessels

For data acquisition, we are using *rotational angiography* [1, 2], where up to 140 X-ray-based projections are taken from a rotation range of 200 degrees around the patients. Based on these individual projections, a 3D volume is reconstructed which represents the respective blood vessels (if a contrast agent is injected) and other anatomical structures in very high resolution. However, the relatively short scanning times of up to 13 seconds make rotational angiography still too slow for fast movements (ie., heart beat).

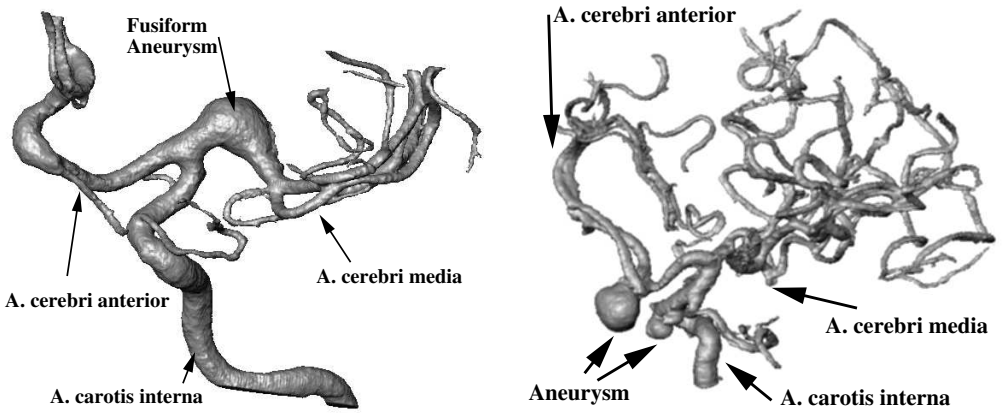
A common procedure in neuroradiology is the examination of extra- and intracranial blood vessels. The major motivation behind these examinations is the diagnosis of cerebral aneurysms. Clinically, two major forms of aneurysms are distinguished; the *fusiform aneurysm* and the *non-fusiform* or *other aneurysms*. The fusiform aneurysm is an expansion of an arterial blood vessel through all of its wall layers (see Fig. 1a). The basic criterion is that no *neck of the aneurysm* can be determined which renders the aneurysm effectively as non-treatable. In contrast, a neck or exit can be identified for *non-fusiform aneurysm* (see Fig. 1b). From an anatomic point of view, other distinctions are possible according to the shape and location of the aneurysms. Furthermore, some aneurysms include a rupture of the inner arterial wall layers, the intima and media. The remaining adventitia layer forms a saccular deformation which is very sensitive to pressure. However, these differences cannot be easily identified which results in no clinical relevance of this distinction.

The expansion of the aneurysms can introduce pressure on other blood vessels – possibly resulting in the occlusion of that blood vessel –, or on surrounding commissures (nerve fibers). This pressure can result in severe headache, partial paralysis, or a stroke. Furthermore, strong blood flow vortices and swirls at the neck and in the aneurysms increase the risk of a highly dangerous rupture of the artery, in particular if the blood pressure is increasing due to physical exercises. This rupture in turn results in the serious destruction or necrosis of the surround brain tissue or in a lethal mass bleeding.

The usual procedures to treat aneurysms include neurosurgical and neuroradiological interventions. The major neurosurgical procedure is the exclusion of the aneurysm from the blood flow by positioning a clip on the neck of the aneurysm. In neuroradiology, tiny platinum spirals ("coils") are introduced into the dome of the aneurysm using a micro-catheter, which is usually inserted via one of the femoral arteries of the legs. Together with the clotting of the thrombocytes, the coils close the aneurysm from the blood stream. All these interventions require the identification of the neck and exit of the aneurysms. However, these tasks can frequently not be achieved with standard 2D angiographies, MIPs, or slicing through the volume dataset. 3D geometry reconstructions using direct or indirect volume rendering techniques[3] have been recently introduced into the clinical practice of research hospitals[2] to provide a better understanding of the angio-architecture of aneurysms in complex blood vessel trees. In particular endovascular inspections of the blood vessel can provide valuable information on the position, orientation, and connection of the aneurysm [4].

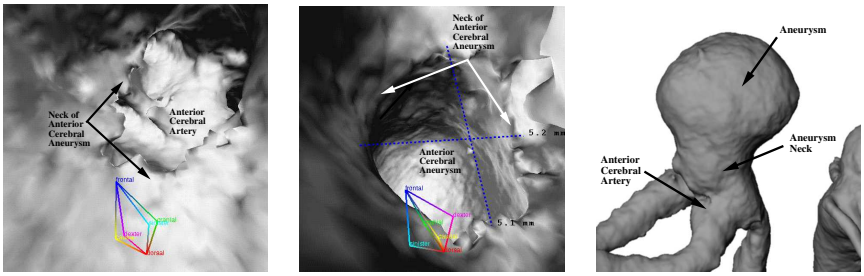
Angioscopy of Coronary Blood Vessels

The heart of the human body is responsible for circulating the blood through the blood vessels of the



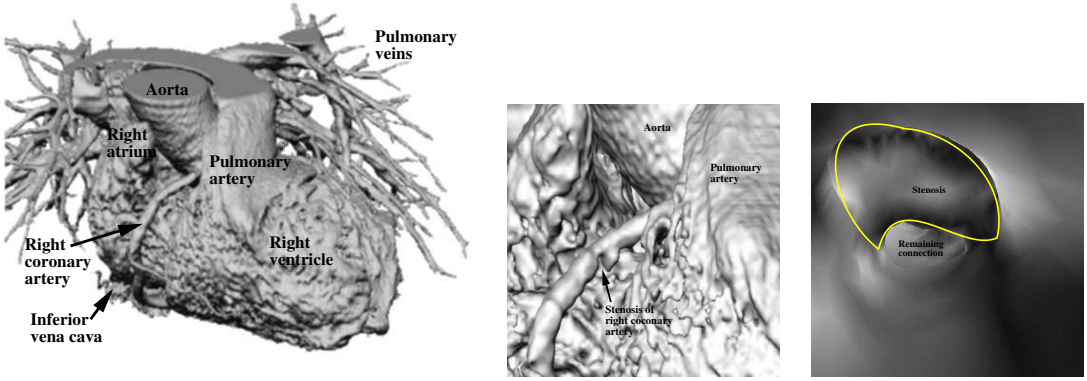
(a) (b)

Figure 1: Cerebral aneurysms; (a) fusiform aneurysm, (b) non-fusiform aneurysm.



(a) (b) (c)

Figure 2: Non-fusiform aneurysm – (a) View from anterior cerebral aneurysm into anterior cerebral artery. (b) View from anterior cerebral artery into aneurysm. Measurements show the size of the aneurysm neck. (c) Anterior cerebral artery from outside.



(a) (b) (c)

Figure 3: (a) Front/top view on Contrast media filled cavities of the heart[5], (b) Outside reconstruction of stenosis of right coronary artery; (c) endoscopic view on stenosis

human body. It is organized into four chambers, the atria and ventricles of the heart (see Fig. 3a). One atrium and one ventricles belong to the *right heart*, or *left heart* respectively. The heart is connected with the blood vessel system through veins (leading towards the heart) and arteries (leading away

from the heart). The venous blood from the various body parts arrives in the *right atrium* through the *great veins*, the *superior* and *inferior vena cava*. It enters through the right *atrioventricular valve*, the *tricuspid valve*, into the *right ventricle*, where it is pumped via the *pulmonary valve* into the *pulmonary artery* to the lungs. After the refreshing of the blood with oxygen in the lungs, it arrives through the *pulmonary veins* in the left atrium. It passed via the *left atrioventricular valve*, the *mitral valve*, into the *left ventricle* and is pumped via the *aortic valve* into the *aorta*, which distributes the blood to all body parts. Directly after the aortic valve are the entrances to the *left* and *right coronary arteries*, which supply the heart muscle with blood. The left coronary artery bifurcates into the *anterior interventricular branch* and the *circumflex branch*, which supplies most of the left heart, and it is the main source of supply of the interventricular septum – which separates the left and right ventricles –, that includes most of the conducting system of the heart. The right coronary artery extends over the right heart. It supplies this part of the heart, the interatrial septum – which separates the left and right atria –, and additionally the interventricular septum, including the sinuatrial and atrioventricular nodes, which are the major parts of the conduction system of the heart [6].

Cardiac diseases are among the number one causes of life-threatening diseases in Europe and North-America. Usually, the under-supply of the heart muscle with oxygen through the blood leads to severe arrhythmia. This arrhythmia can cause an electro-mechanic decoupling of the conduction of the heart, resulting in a dangerous reduction of the pumping performance, in a possible collapse of the blood circulation, and finally the death of the patient. Even if the arrhythmia is not leading to a fatal decoupling of the conduction, the missing supply of oxygen leads to the necrosis of that part of the heart muscle, if necessary medical procedure are not applied in time. The actual cause of the under-supply of the heart muscle is usually a stenosis or an occlusion of a coronary artery. Other reasons for arrhythmias are direct injuries of the heart muscle by force, failure of the valves of the heart, or infectious diseases.

For the diagnosis of these malfunctions, several clinical and laboratory tests are applied. For imaging, echo cardiography, coronary angiography, or nuclear medicine methods are used. However, only 3D scanning methods are able to generate volumetric data of the heart. Unfortunately, most modalities are too slow to avoid motion artifacts of the fast moving heart. More or less only ECG-triggered (Electro-Cardio-Gramme) spiral CT provides scanning which captures volumetric data of the heart that is widely motion artifact free. With the introduction of multi-slice CT, enough spatial resolution is also available for 3D coronary angiography.

Based on an anisotropic dataset from multi-slice CT coronary angiography, we explore the use of virtual endoscopy for virtual angioscopy of the heart[5]. The dataset consist of 150 slices at a slice distance of 1.25mm. Each of the slices has a resolution of 512×512 pixels at a pixel distance of 0.59 mm. Outside views on the reconstructed, contrast media filled cavities of the heart can be seen in Figure 3a. The reconstructed geometry of the heart is very complex. From more than two million polygons, more than 80% of the geometry was culled by our visibility driven rendering, thus obtaining a frame-rate with an average of 4.2 fps on a typical path through the right heart. The low culling and frame-rate are a result of the deep visibility within the chambers of the heart which reduces possible culling benefits.

The coronary valves, which open and close at a high speed, cannot be reconstructed completely, due to motion artifacts in the scanned dataset. In particular the right atrium of the heart is subject to artifacts which are due to the injection of the contrast agent and they can be seen in the respective slices of the CT dataset. However, other important features of the anatomy of the heart of the patient are visible, such as the aorta, the pulmonary artery, the great veins, the pulmonary veins, and the coronary arteries. Of special interest are the latter one's, since most of the cardiac emergencies result from a stenosis of these arteries. Figure 3b and c shows such a stenosis of the right coronary artery, which supplies the right heart and important parts of the conduction system of the heart. Measurements of the diameter of the blood vessel at of the stenosis provide a quantitative evaluation of the stenosis. Additionally, measurements of the volume of the ventricles provide information of the performance of the heart.

Discussion

The different objectives of virtual endoscopy applications impose specific requirements on the virtual endoscopy system. An educational objective focuses more on the visual quality that demonstrate the general topological and geometric aspects of the specific patient anatomy. In contrast, the accuracy of fine details is only of limited importance, if the details are not the subject of the examination. However, this is different for a clinical objective where the accurate rendering is one of the major factors that determine the usability, where

an incorrectly represented blood vessel connection might have a fatal impact on the medical intervention. If virtual endoscopy is used for planning an intervention, it is important that relevant anatomical structures are represented appropriately, since otherwise the planned access path (ie., in virtual ventriculoscopy) might be occluded in the "real world" anatomy. Similar, the visual representation must be highly accurate for intra-operative navigation to provide usable information to the surgeon.

There are several sources of errors that can lead to an inaccurate visual representation of anatomical structures in virtual endoscopy. Most notorious are partial volume effects and undersampling which generate "fake" connections between the various caverns that are actually not connected. Furthermore, motion artifacts can reduce the visual quality severely or distort the actual anatomical geometry during a (long) medical scanning procedure or scans of fast moving organs (ie., the heart).

Finally, for all procedures that require a histological examination of a tissue sample under a microscope, virtual endoscopy is not able to compete. The data resolution of modern 3D scanners does not reach into the resolution of a microscope, although it is already in a sub-millimeter range. Furthermore, texture information - such as structure, color, and reflections - is also not captured by 3D scanners. Similarly, if the medical procedure includes the removal of tissue (ie., lesions or tumors), or other objects, invasive or minimally invasive procedures cannot be replaced by virtual endoscopy, since it does not interact with the actual body of a patient.

Acknowledgements

This work has been supported by the German Federal Ministry of Research and Education, by the State of Baden-Württemberg, by the Hewlett-Packard Corporation, by DFG project CatTrain, and by the Competence Center of minimally invasive medical technology Tübingen-Tuttlingen.

I like to thank my collaborators in the many projects, namely Özlem Gürvit, now with the University Hospital of Marburg, Dirk Freudenstein, Jürgen Hoffmann, Dirk Troitzsch, Martin Skalej, Andreas Bode, Andreas Kopp, Florian Dammann, and Claus Claussen of the University Hospital Tübingen, and Dirk Mayer of the University Hospital Mainz. In particular, I like to thank the members of the VCM group in Tübingen: Anxo del Río, Jan Fischer, Jasmina Orman, and Zein Salah.

References

[1] R. Fahrig. *Computed Rotational Angiography*. PhD thesis, University of Western Ontario, 1999.

[2] Ö. Gürvit, M. Skalej, R. Riekmann, U. Ernemann, and K. Voigt. Rotational Angiography and 3D Reconstruction in Neuroradiology. *electro medica*, 68(1):31–37, 2000.

[3] D. Bartz and M. Meißner. Voxels versus Polygons: A Comparative Approach for Volume Graphics. In *Proc. of Volume Graphics*, pages 33–48, 1999.

[4] B. Marro, D. Galanaud, C. Valery, A. Zouaoui, A. Biondi, A. Casasco, M. Sahel, and C. Marsault. Intracranial Aneurysm: Inner View and Neck Identification with CT Angiography Virtual Endoscopy. *Journal of Computer Assisted Tomography*, 21(4):587–589, 1997.

[5] D. Bartz, Ö. Gürvit, M. Lanzendörfer, A. Kopp, A. Küttner, and W. Straßer. Virtual Endoscopy for Cardio Vascular Exploration. In *Proc. of Computer Assisted Radiology and Surgery*, pages 960–964, 2001.

[6] W. Hollinshead and C. Rosse. *Textbook of Anatomy*. Harper & Row, Publishers, Philadelphia, USA, 4th edition, 1985.

Visualisering av en berg- og dalbane

Hans Birger Drange og Harald Soleim, Høgskolen i Bergen

Studentene i kurset SOD165 Grafiske metoder ved dataingeniørutdanningen, Høgskolen i Bergen, visualiserte en berg og dalbane i sin siste obligatoriske øving. Studentene får selv foreslå oppgaver. De forslag som kom inn og som vi gjennomførte var (1) Visualisering av western-duell og (2) visualisering av en berg og dalbane som er presentert nedenfor.

Krav til oppgaven var at den skulle følge fysiske lover gitt i oppgaveteksten.

Problembeskrivelse

I denne oppgaven skal vi visualisere en vogn som går i en berg- og dalbane. Vi betrakter først en kurve/bane som går i et vertikalt plan, (I), og så en kurve/bane som utgjør en spiral (II). Kjøretøyet tenkes å gå på skinner uten friksjon og luftmotstand.

(I) Bane i vertikalplan:

Vi har laget fig.1 med tanke på at vognen i denne situasjonen har mulighet til å forlate banen. $\vec{N}$ er normalkraften som virker fra banen på vognen og har retning langs krumningsradius.

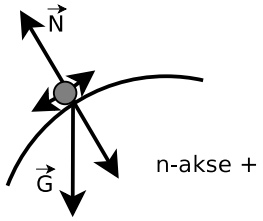


fig.1

G_n er tyngdens komponent i n-retningen, og N_n er normalkraften sin n-komponent. Newton's annen lov gir oss da:

$$N_n + G_n = m\frac{v^2}{r} \tag{1}$$

der m er vognens masse, v er farten og r er krumningsradiusen. Hvis N_n blir positiv beregnet fra (1), betyr det at vognen har mistet kontakt.

Vi gjør tilnærmete beregninger av vognens bane ved å dele opp tiden i like små tidsluker. La oss se på en vilkårlig tidsluke: $[t_0, t_1]$. Veien (buelengden) i dette tidsrommet beregnes da ved:

$$s = \frac{1}{2}a_0(t_1 - t_0)^2 + v_0(t_1 - t_0) \tag{2}$$

og farten ved $v = a_0 t + v_0$. Når a_0 og v_0 er kjent, så kan da s og v beregnes.

Når vi kjenner buelengden, kan vi beregne punktet i banen for tiden t_1 ved å bruke en parameterfremstilling for banen $\vec{r} = \vec{r}(\tau)$, hvor τ er parameteren.

Da blir $s = \int_{\tau_0}^{\tau_1} |r'(\tau)|\, d\tau$ og vi setter $r'(\tau) \approx r'(\tau_0)$. (idet alle størrelser ved starten av tidsluken er kjent, τ_0 er her parameteren for starten av tidsluken). Da får vi

$$s = |r'(\tau_0)|\,(\tau_1 - \tau_0) \tag{3}$$

Og siden s er kjent fra (2), så finner vi her τ_1, parameteren ved slutten av tidsluken. Dermed kjenner vi også punktet i banen ved slutten av tidsluken fordi parameterfremstillingen er kjent.

Beregningene våre har nå forutsatt at vi kjenner a_0 og v_0 ved starten av tidsluken. Farten v_0 antas kjent ved starten av hele bevegelsen og a kan også beregnes ved start ut fra retningen ved start og tyngden. Baneakselerasjonen a blir jo lik tyngdens akselerasjon i bevegelsesretningen. Dermed er disse kjent for første tidsluke og siden vil det fortsette av seg selv.

(II) Spiral:

La oss f.eks. velge en spiral med parameterfremstilling:

$$\vec{r}(\tau) = b\cos\omega\tau\vec{i} + b\sin\omega\tau\vec{j} + \lambda\tau\vec{k}$$

der b, ω og λ er positive konstanter. (Dette gir en spiral med bevegelse oppover "mot urviser". Tilsvarende analyse kan gjøres for bevegelse nedover.)

Newtons 2. lov i bevegelsesretningen sier da som i sted at baneakselerasjonen er lik tyngdens akselerasjon i bevegelsesretningen.

Siden kurvenormalen er horisontal, så har tyngden ingen virkning i normalretningen, og Newtons 2. lov i normalretningen sier da bare at $N_n = m\frac{v^2}{r}$, og er en informasjon som vi ikke bruker i det følgende. Newtons 2. lov i retningen av kurvens binormal blir slik:

$$0 = G_b + B_b \tag{4}$$

hvor G_b er tyngdens komponent langs binormalretningen og B_b er kraften fra banen i binormalretningen.

Skal banen kunne virke med kraften B_b, så må denne være positiv. Og siden $G_b < 0$, (fordi

binormalretningen "peker oppover"), så blir B_b beregnet fra vår ligning positiv siden $B_b = -G_b$, og dermed blir denne ligningen også automatisk oppfylt hele tiden og får ikke innvirkning på det vi er interessert i å beregne.

Beregningen av punktene i banen etter som tiden går, gjøres som under punkt (I), men nå får vi at integralet i formelen $s = \int_{\tau_0}^{\tau_1} |r'(\tau)|\, d\tau$ kan regnes ut eksakt og vi får

$$s = \sqrt{\omega^2 b^2 + \lambda^2}(\tau_1 - \tau_0) \qquad (5)$$

istedet for (3).

Alternativt kan en bruke bézierkurve til å lage banen.

BMK. En undersøkelse som vi ikke har gjort når det gjelder spørsmålet om vognen kan holde seg i banen, er om den kan velte når den kjører i spiralen. Men da vil form og massefordeling i vognen spille en rolle og dermed ville gjennomføringen av programmet bli unødig komplisert.

Løsningsforslag

En gruppe studenter benyttet 8 béziersegmenter til å lage selve banen. Banen er plassert i en stor kube eller "skybox" hvor det på innsiden er lagt en tekstur for å få inntrykk av at vognene beveger seg i et landskap. Det er laget eget kamera for banen slik at vi kan følge vognen rundt banen. Studentene ble stilt relativt fritt til å velge matematisk algoritme.
Tegning i aktuell tidsluke:

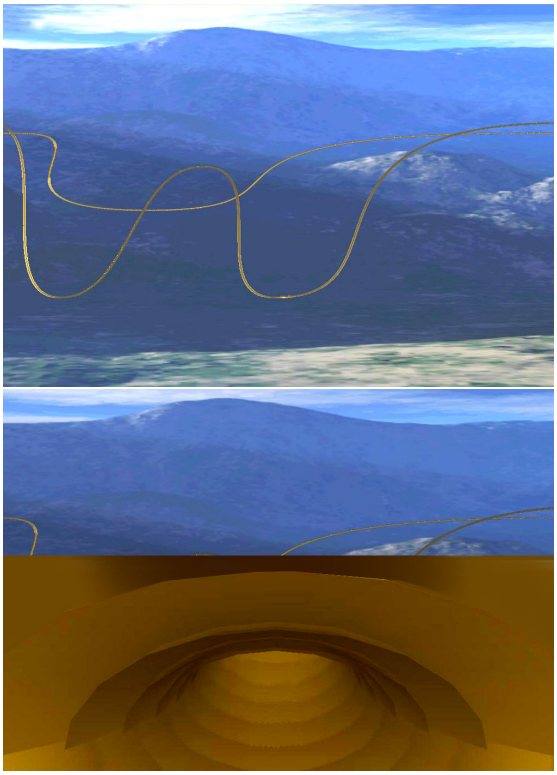
- Programmet finner punkt for vognens posisjon.
- Finner vinkler for vognen.
- Nå kan vognen tegnes.

Forberedning av data for neste tidsluke.

- Finner veistykke.
- Beregner akselerasjon.
- Beregner ny fart for neste tidsluke.
- Beregner ny posisjon (kurveparameter) til startpunkt for neste tidsluke.

Det er ikke tatt hensyn til friksjon og luftmotstand.

Studentgruppen bestod av: Steinar Henriksen, Thomas Johansen, Kjetil Solheim og Frode Peder Årvik.



Modellering av Turbidittar

Ragnar Hauge, Norsk Regnesentral

Visualisering av store 3D-modellar av oljereservoar er viktig i oljeindustrien, og noko det vert satsa mykje pengar på, mellom anna med Hydro sin CAVE. Imidlertid må ein først ha ein modell som kan visualiserast og eit verktøy for å generera desse modellane. Vi i SAND-gruppa ved Norsk Regnesentral har lenge jobba med slike modellar. No er vi, i samarbeid med Universitetet i Bergen i gang med eit større prosjekt på turbidittmodellering. Det er finansiert av Forskningsrådet, ConocoPhillips og Hydro.

Turbidittar er spesielle sandsteinsformasjonar som vert avsett på djupt vatn. Sandstein dannar gode oljereservoar, sidan den er svært porøs, og dermed kan innehalda mykje olje. Sandstein vert danna frå sediment som ligg nedgrave over lang tid.

Turbidittavsetjinga består av store undersjøiske sedimentras. Desse går utanfor elvemunningar, der det vert avsett store sedimentmengder. Desse avsetjingane vert ustabile og rasar ut, og summen av mange slike ras utgjer eit turbidittreservoar. Mellom kvart ras skjer det ei konstant utfelling av fine partiklar frå vatnet (opphavleg frå elva), som dannar leire. Leiren er kompakt, og bidrar ikkje til reservoaret.

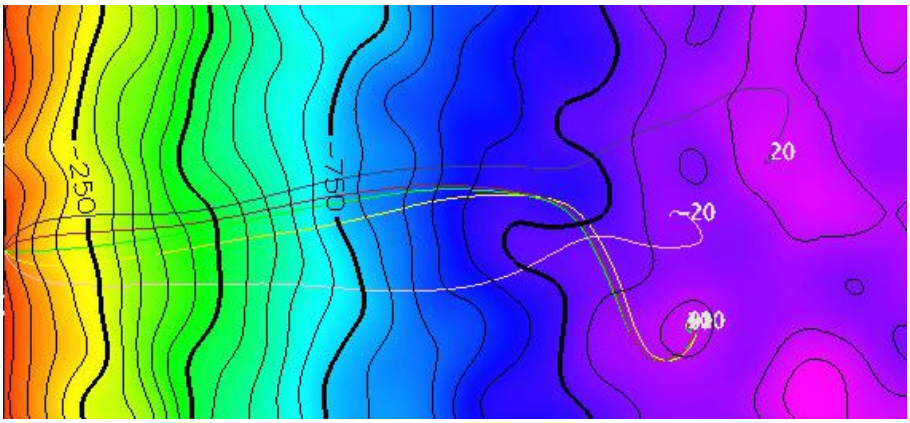
Avsetjingsmekanismen gir turbidittreservoar karakteristiske eigenskapar. Til tross for at rasa går i svært flate områder, med helning på under 4 grader, er dei energirike. Initielt vil derfor eit slikt ras grava med seg litt av botnen, som består av tidlegare avsetjingar. Dette fører til at det i det "bratte" området vert danna ein kanal, slik at mange ras vil følgja same løp her.

Når raset når flatare område og byrjar å tapa energi slår andre mekanismar inn. Det vil då følgja konturane i botnen betre, noko som gjer at kvart ras går sin eigen veg. Dette, i tillegg til at raset naturleg vidar seg ut når det sakkar ned og kjem på flatmark gir slike reservoar ei karakteristisk vifteform.

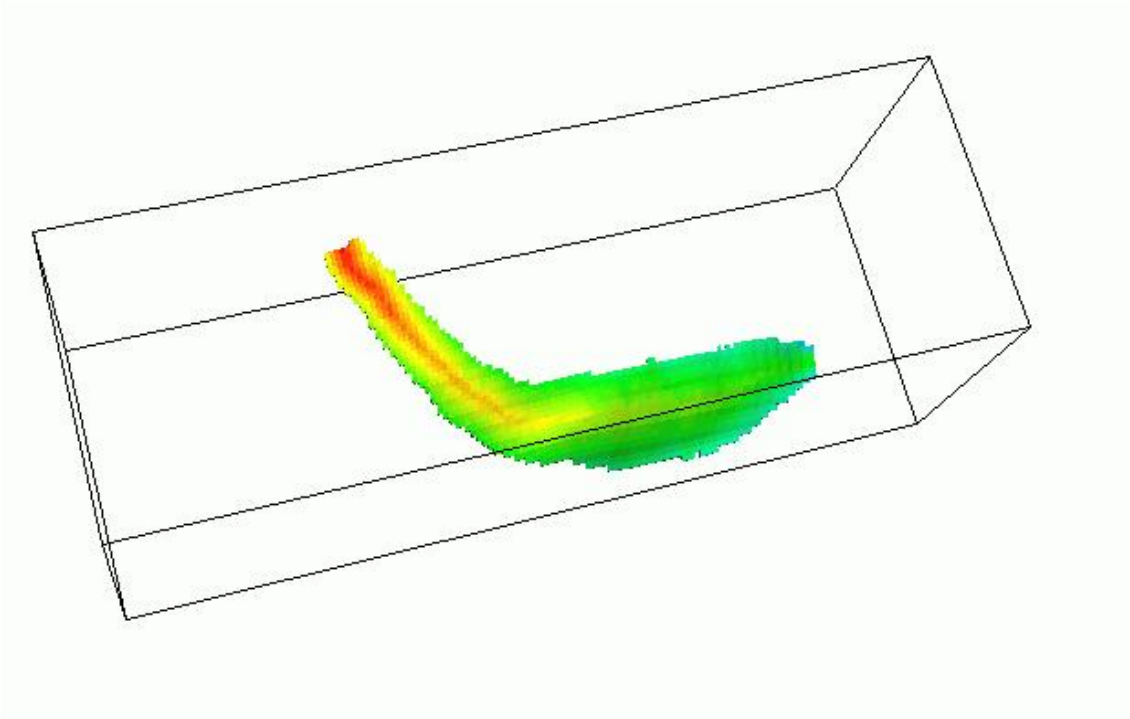
Planen i dette prosjektet er å nytta ein forenkla fysisk modell av prosessen til å bygga reservoaret. Forenklinga gjer det mulig å prøva mange ulike ras med ulike initialvilkår i kvart steg, sidan det vil gå raskt å generera kvart ras. I tillegg vert det lagt inn stokastiske ledd i prosessen som hjelper med databetinginga. Dette gir etter planen ein realistisk turbidittmodell som kan brukast for utvinning.

Ettersom geologisk modellering er svært visuelt i sin natur, er visualisering av resultatet viktig både for kvalitetssikring av modellen og for å forstå kva den predikerer. Eit problem i denne samanhengen er at våre modellar er stokastiske, slik at kvar realisasjon er eit mulig utfall som stemmer med modell og data. Dette er egna til å gi kontroll på usikkerheten, men usikkerhet er vanskeleg å visualisera.

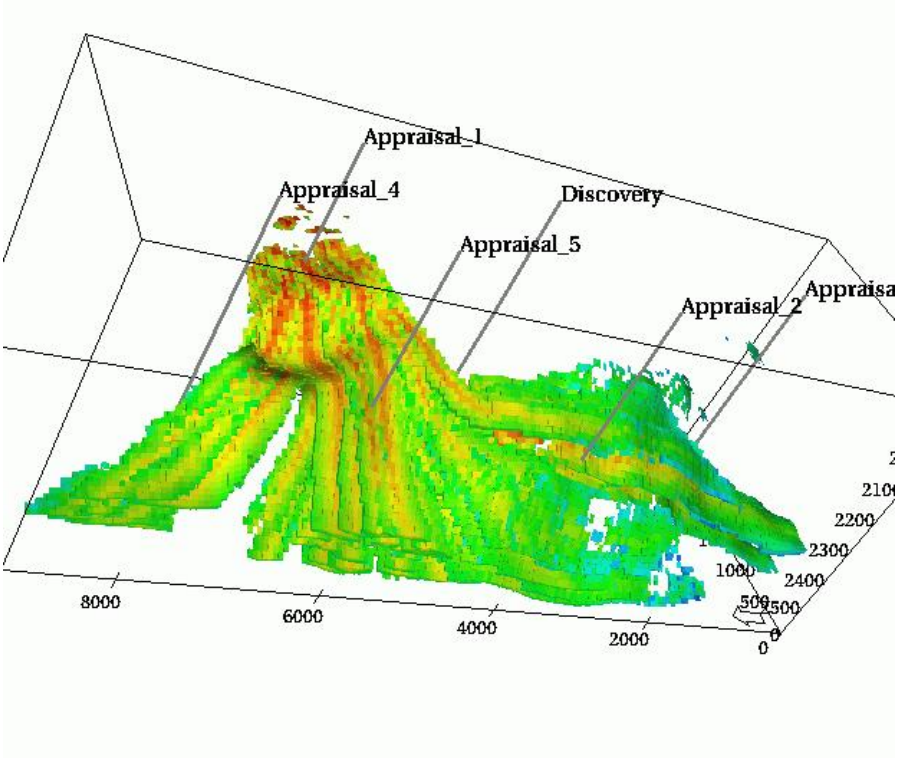
Merk at realisasjonane vist her ikkje er frå dette prosjektet, men frå eksisterande turbidittmodellar.



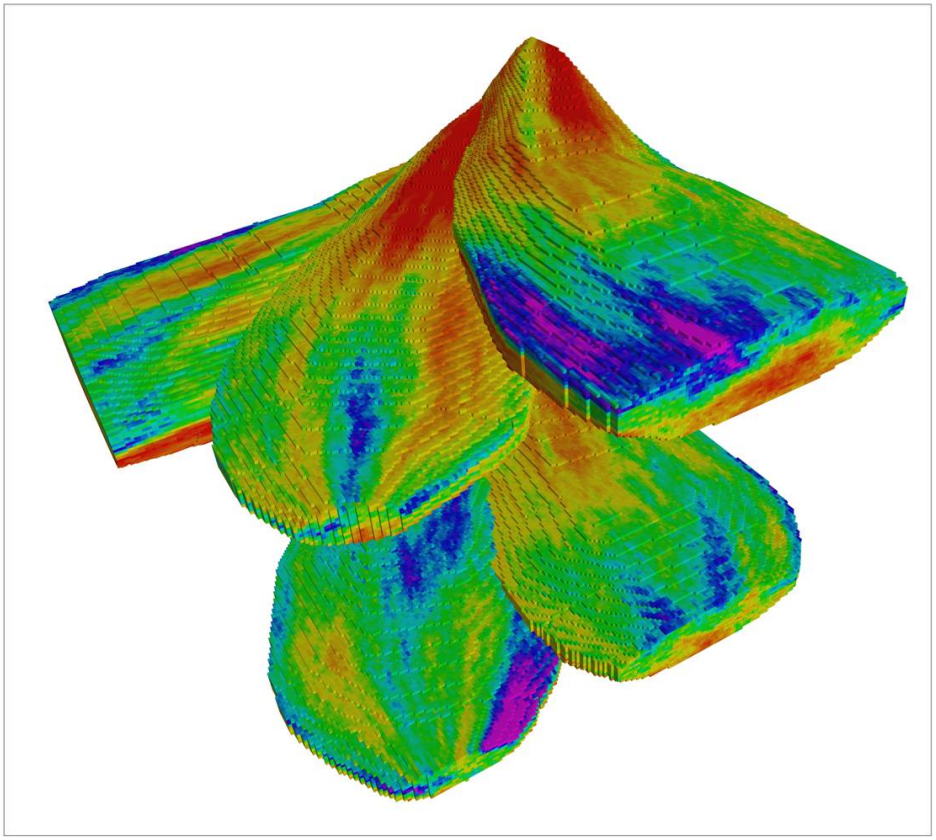
Mulige baner for eit trubidittras. Fargen viser djup, med raud som grunn og lilla som djup.



Eksempel av eit turbidittras med innlagt porøsitet som fargekode.



Full realisasjon av trubidittreservoar, med porøsitet som fargekode.



Utsnitt frå realisasjon av turbidittreservoar, med porøsitet som fargekode.

Turbidittavsetjing sett i ei skjæring i Pyreneane.

Grafikkort som regneressurs

Tor Dokken, Knut–Andreas Lie, SINTEF IKT, Avdeling for Anvendt matematikk

Våren 2003 ble det introdusert grafikkort med 32-bits flyttallspresisjon og multighet for å programmere fragmentprosessoren i høynivåspråk. Dette åpnet opp for en rekke nye datagrafiske muligheter, da en programmerer ikke lenger må forholde seg til gitte operasjoner i en fiksert pipeline. En annen spennende mulighet som har åpnet seg er bruk av grafikkortet som regneressurs. Mens majoriteten av transistorene i en CPU inngår i cache-strukturen, brukes majoriteten av transistorene i et moderne grafikkort til flyttallsprosessering. Med 16 parallele pipelines som hver jobber simultant på vektorer av lengde fire (tilsvarende RGBa), sitter det en liten superdatamaskin gjemt under panseret i mange nye PCer!

Helt siden datamaskinens barndom har datagrafikk vært et viktig verktøy for presentasjon av resultater. For noen anvendelser er CPU-ressurser tilstrekkelig for å produsere datagrafikk, mens andre anvendelser krever hardwareakselerasjon. I løpet av de siste årene har 3D datagrafikk blitt en viktig komponent i mange dataspill. Dette har skapt et stort marked for kraftige grafikkort. I dag er programmerbare grafikkort blitt svært rimelige; for PC er de priset mellom 1500 kr og 4000 kr.

Ved SINTEF IKT, avdeling for Anvendt matematikk, har 3D grafikk i mange år vært et viktig verktøy for vår forskningsaktivitet innen simulering og datastøttet geometri. Da grafikkort med programmerbare grafikk prosessorer (GPUer) ble introdusert, innså vi raskt at disse kunne bli en viktig regneressurs både innen datastøttet geometri og simulering (partielle differensialligninger). I 2003 søkte vi Norges forskningsråd om et strategisk instituttprosjekt (SIP). Søknaden lyktes, og prosjektet *"Grafikkort som høyytelses regneressurs"* (2004-2007) ble startet. Mer informasjon om prosjektet finnes på følgende url: `http://www.math.sintef.no/gpu/`.

Vi er nå i prosjektets annet år, og vår kunnskap om GPU-programmering blir stadig bedre. En av våre ambisjoner er å spre denne kunnskapen også utenfor datagrafikkens rekker, og vi håper at denne artikkelen er et skritt i riktig retning.

Noen prosjektresultater

En viktig del av prosjektet så langt har omhandlet numerisk løsning av partielle differensialligninger ved hjelp av grafikkort. Vi har studert en rekke differensialligninger, fra enkle lineære ligninger som varmeligningen og bølgeligningen, til ikkelineære systemer av bevarelseslover (Buckley-Leverett ligningene for tofasestrøm i porøse medier, Saint-Venant ligningene for bølger på grunt vann og Euler-ligningene for kompressible gassdynamikk). Mens forskere ellers rundt om i verden i stor grad har vært opptatt av implisitte metoder og numerisk lineær algebra, har vi i hovedsak fokusert på tidsavhengige problemer med eksplisitt tidsintegrasjon. Dette fordi eksplisitte metoder er svært enkle å parallellisere og derfor har stort potensial for å kunne utnytte paralleliteten i GPUer.

Så langt har vi god erfaring med bruk av GPUer til regneressurs. For de mest komplekse anvendelsene vi har studert, har vi observert en hastighetsøkning for store grid på mellom 10 og 20 ganger sammenliknet med tilsvarende kjøringen på en kraftig CPU. For enkle ligninger og enkle numeriske metoder ligger ytelses-forberingen mer rundt en faktor 2-5. Dette kan kanskje virke noe rart, men forklaringen er enkel. For skjema med få flyttallsoperasjoner er forhåndshenting av data ikke rask nok til å utnytte regnekraften maksimalt. For mer komplekse skjema, utføres derimot flere beregninger per hentet dataenhet og dermed utnyttes hardwaren bedre. En enkel tommelfingerregel sier at man bør utføre 6-8 operasjoner per hentet dataenhet for å utnytte ytelsespotensialet for GPUene. Tilsvarende, har vi sett at for små beregningsgrid reduserer overhead knyttet til å etablere pipeline og initialisere shadere ytelsesforbedringen (shader = program på GPU). Den eneste store begrensningen vi har støtt på så langt, er tilgjengelig antall hardwareregistre. Noen av de skjemaene vi har sett på har krevd mer mellomlagring enn vi har hatt tilgjengelige registre. Vi har derfor i et par tilfeller vært nødt til å reberegne mellomresultater, noe som har redusert den observerte ytelsesforbedringen.

En annen anvendelse vi har sett på er bruk av GPU som en akselerator i CAD-type skjæringsproblemer (skjæring mellom flater i CAD-modeller), hvor vi har utviklet en metodikk som vi har søkt patent på. I skjæringsalgoritmer kan fragmentprosessoren brukes for oppdeling av flatene i biter for å skape nøyaktige innhyllinger av flatene. Vi kan også skape nøyaktige innhyllinger for flatenes felt av normaler.

- GPUens dybdebuffer kan brukes for å teste

om to flaters innhyllninger har overlapp. Hvis innhyllningen ikke overlapper så kan heller ikke flatene skjære hverandre.
- For at to flater skal skjære hverandre i en lukket kurve, må flatenes felt av normaler overlappe. Ved å teste om innhyllningen av to flaters felt av normaler overlapper, kan en avgjøre om to flater kan skjære hverandre i en lukket kurve.
- De overnevnte shadere kan også finne informasjon om hvilke områder som overlapper. Dette kan hjelpe de delene av skjæringsalgoritmen som løper på CPUen til å arbeide raskere.

Høynivåspråk og drivere

På mikroprosessorer kompileres programmer i høynivåspråk til assemblykode, som er nært beslektet med instruksjonssettet til den den aktuelle familie av mikroprosessorer. Et slikt instruksjonssett er gyldig i mange år og kan bli utvidet i senere generasjoner av mikroprosessorfamilien.

GPUer programmeres i høynivåspråk, og GPU-produsentene leverer drivere som overfører shaderkode til hardwareoperasjoner. Det finnes assembly-lignende språk for ulike GPU-familier, men disse anbefales ikke for programmering. Assemblykode er imidlertid et godt verktøy for å forstå hvordan kompilert shaderkode benytter hardwareressurser. Ved å studere den assembly-lignende koden som lages kan man finne ut hvordan en fragmentshader kan optimaliseres for å utnytte hardwareressurser bedre. Et eksempel på dette er å bruke assemblykoden til å finne ut hvordan man kan skrive om shaderkoden slik at den benytter færre hardware-registre.

For GPUer finnes det populære platform- og systemuavhengige APIer: DirectX og OpenGL. Begge har høynivåspråk for shader programmering.

- HSL (DirectX High-Level Shading Language), et C-likt språk som inngår i DirectX fra Microsoft. Språket er utviklet i samarbeid med nVidia. Microsoft påvirker GPU-utviklingen ved å kreve at framtidige GPUer er kompatible med DirectX.
- GLSL (OpenGL Shading Language), et C-likt språk som er uavhengig av operativsystem. De store GPU-produsentene gjør ny funksjonalitet tilgjengelig gjennom egne OpenGL utvidelser, og foreslår disse som utvidelser av OpenGL standarden.
- Cg (C for graphics), et annet C-likt språk utviklet av nVidia for nVidia GPUer. Cg er uavhengig av grafisk API, og driverne kan generere shaderkode både for OpenGL og DirectX. Denne uavhengigheten har medvirket til et høyere abstraksjonsnivå i språket.

De programmerbare delene av grafikk "pipeline"

I nyere GPU er det to programmerbare deler:

- Vertexprosessoren kjører programmer som kalles vertexshadere.
- Fragmentprosessoren kjører programmer som kalles fragmentshadere.

Den grunnleggende datatypen til begge typer shadere er vektorer av lengde 4 og 4×4 matriser begge med 32-bits flyttallspresisjon. Standard operasjoner på disse er effektivt implementert i hardware. Avhengig av GPU-serie kan mange andre operasjoner også være hardwareakselerert, f.eks. elementvise logaritmiske og trigonometriske funksjoner. En av de store fordelene med slike funksjoner er at de tillater parallell eksekvering, fordi resultatet i et vektorelement er uavhengig av resultatene i andre vektorelementer.

Fragmentprosessoren

Mye av regnekraften til GPUen er lagt til fragmentprosessoren. I fragmentprosessoren til f.eks GeForce 6-serien som ble lansert av nVidia i 2004, er det opptil 16 pipelines. Hver av disse pipelinene kan utføre 4 flyttalsoperasjoner i parallell, med en maksimal hastighet på 450MHz. Syntetiske tester (programmer som er laget for å demonstrere ytelse) viser en total regnekraft på 53GFlops. Det er forventet at regnekraften vil fortsette å øke i tiden framover ved at grafikkortene får flere pipelines og at klokkehastigheten økes. Dette gjelder både GPUer fra begge de store produsenten, nVidia og ATI.

Nye GPUer støtter dynamisk forgrening i både vertex- og fragmentshadere. De enkelte pipelines i fragmentprosessoren arbeider synkront ved at de utfører samme gren. I en "if/else" struktur risikerer man derfor at at begge grener utføres, og at det dermed ikke spares tid. Håndteringen av "if/else" strukturer er avhengig av GPU-generasjon. Det forventes forandringer i fremtidige GPU-generasjoner. Det kan oppstå flaskehalser ved dynamisk forgrening i shadere, men disse kan en søke å unngå som følger:

- I mange tilfeller kan shadere skrives om til å bruke aritmetiske uttrykk i stedet for dynamisk forgrening. Dette er spesielt nyttig når vektoregenskapene til GPUen skal benyttes.

- Tidlig utgang er en mulighet for å unngå å eksekvere delere av en pipeline. Et grafikk-relatert eksempel er lysberegninger, hvor man kan benytte tidlig utgang for å unngå å beregne en kompleks lysmodell for punkter som ligger langt fra lyskilden.

Disse strategier gjelder for nåværende GPU-generasjon, men forventes å være aktuelle også i framtidige generasjoner.

Programmering av fragmentprosessoren

For å forstå GPU-programmering er det viktig å forstå dataflyten i den grafiske pipelinen. Figur 1 viser en forenklet versjon av en grafiske pipeline:

- Først må geometrien defineres og informasjonen som skal prosesseres må kobles til geometrien.
- Geometristrukturene settes sammen til grafiske primitiver. De grafiske primitivene klippes for å fjerne deler som ligger utenfor synsvolumet.
- De synlige deler av primitivene rasteriseres i samsvar med oppløsningen til "render target". I denne prosessen genereres det fragmenter som skal bearbeides av fragmentprosessoren.
- Så kjøres fragmentshaderen på hver fragment.

For å forstå GPU-programmering er det viktig å forstå rollen til teksturer. En tekstur kan sees på som et rektangulært bilde med $n \times m$ punkter, hvor hvert punkt har en til fire verdier. Grafikkort tilbyr to måter for adressering av teksturer, enten gjennom heltallsindekser eller ved normaliserte teksturkoordinater. For å unngå å kjenne de eksakte dimensjonene til en tekstur når en leser data fra den, benyttes normaliserte teksturkoordinater i enhets-kvadratet $[0, 1] \times [0, 1]$.

Oppsett for beregning på GPU

For å kjøre programmer rettet mot ikke-grafiske formål på fragmentprosessoren, må det defineres en geometri som sørger for at de korrekte fragmenter blir skapt. Følgende definisjon sikrer at det blir samsvar mellom (x,y)-koordinatene til geometrien og teksturkoordinatene.

- Definer et rektangel (quad) med hjørner: (0,0,0), (1,0,0), (1,1,0) og (0,1,0).
- Gi hjørnene teksturkoordinatene (0,0), (1,0), (1,1) and (0,1).
- Definer synsvolumet slik at rektangelet akkurat blir synlig.

Dette vil skape fragmenter som alle har teksturkoordinater i rektangelet definert av de fire punktene (0,0), (1,0), (1,1) og (0,1).

Oppsettet over er begrepsmessig enkelt. Når de grafiske primitiver bygges vil imidlertid rektanglet bli brutt opp i to trekanter. GPUen forhåndshenter teksturdata nær det segmentet som skal behandles for å øke beregningshastigheten. Denne forhåndshentingen er imidlertid bare fra det primitivet som rasteriseres. Hvis fragments-haderen trenger aksess til data som ikke er knyttet til primitivet må shaderen vente på disse dataene. Ved å innbake rektangelet i en større trekant kan dette problemet unngås:

- Lag en trekant med hjørner: (0,0,0), (2,0,0), (0,2,0).
- Tildel hjørnene teksturkoordinater (0,0), (2,0), and (0,2).
- Lag et synsvolum som definert over for rektangelet.

Fragmenter utenfor rektangelet (0,0), (1,0), (1,1) og (0,1) vil fjernes av klippesteget i den grafiske pipelinen og det vil kun genereres fragmenter som ligger innenfor rektangelet. Fragmentprosessering vil være assosiert med kun en trekant.

Ved å følge fremgangsmåten som angitt over, kan man skape teksturer og "render targets" som er nært beslektet med "vanlige" arrayer og matriser i C. Forskjellen er at vi i shaderen vil adressere med flyttall, mens C-arrayer bruker heltallsindekser.

Fragmentshadere: leser fra teksturer og skriver til "render targets"

Dagens GPUer kan lese fra mange teksturer, men kan kun skrive til opptil fire "rendert targets". Vi har ikke kontroll over i hvilken rekkefølge fragmentene prosesseres. Dette har den følgen at algoritmer som Gausseliminasjon, der det ferdige resultat og midlertidige data lagres i samme matrise, må implementeres som en serie shadere. Når en ny shader skal startes, må en geometri tilknyttes, og alle skritt i den grafiske pipeline må utføres. Ved starten av 2005 kan f.eks en GPU av typen nVidia GeForce Serie 6 GPU kjøre inntil 8000 shadere i sekundet.

Som vi har påpekt tidligere, må fragments-hadere være beregningstunge for å nyttegjøre seg beregningskaften til fragmentprosessoren. Hvis ikke vil initiering av hver ny shader ta for mye tid, og fragmentshaderen vil gi liten ytelsesgevinst.

Bruk av resultater fra fragmentshadere

Pilene i Figur 1 og 2 illustrerer at resultatet fra en fragmentshader kan bli:

- Kopiert til en vertex array, og senere bli brukt som input til en vertexshader.

- Konvertert til en tekstur og senere bli brukt som input til en fragmentshader.
- Bli gitt tilbake til applikasjonen som kjører på CPUen.
- Visualisert.

Alle shadere som kjører på GPUen må startes av applikasjoner som kjører på CPUen. Data som en shader har produsert, og som skal brukes av andre shadere kan imidlertid forbli på GPUen. Dette medfører at en oppgave som er strukturert i flere shadere ikke trenger å bli synderlig forsinket av båndbredden mellom CPUen og GPUen. Trenger CPUen å prosessere data før neste shader kan starte, vil båndbredden fort bli en flaskehals.

Løkkestrukturer og fragmentprosessoren

Det temmelig enkelt å implementere en dobbel løkke på GPUen, se Figur 2. Kontrollstrukturen for løkken

```
For i=1,?,n;
   For j=1,?,m;
```

erstattes av kall til grafikkpakken. I OpenGL kan dette se ut som:

- Initier viewport

```
glViewport( 0, 0, n, m);
```

- Initier rektangelet

```
glBegin(GL_QUADS);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(0.0f, 0.0f, 0.0f);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(1.0f, 0.0f, 0.0f);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(1.0f, 1.0f, 0.0f);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(0.0f, 1.0f, 0.0f);
glEnd();
```

Definer klippevolumet slik at rektangelet akkurat ligger innenfor. Dette opplegget forsikrer at alle $n \times m$ fragmenter rasteriseres i rektangelet $[0,1] \times [0,1]$. Selve innholdet i løkken vil være fragmentshaderen, inputdata vil leses fra teksturer, og resultater skrives til et eller flere "render targets".

Eksempler på GPU programmer

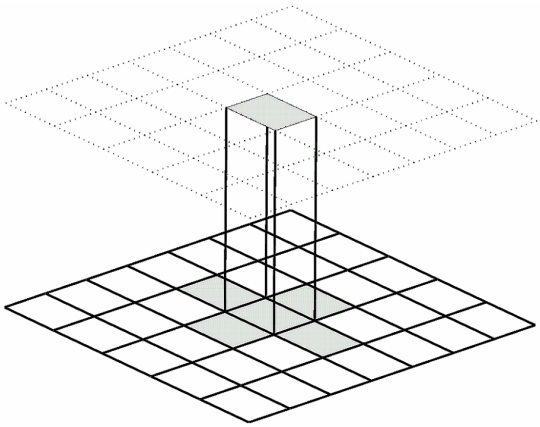
Vi vil nå gi to eksempler på hvor enkle shadere for endelige differansemetoder for løsningen av partielle differensialligninger kan være. Deretter viser vi et eksempel på ett av de mer avanserte problemene vi har løst: flodbølger i et dalsystem generert av et dambrudd.

Varmeligning

Den todimensjonale varmeligningen for transport av varme i et legeme er $u_t = u_{xx} + u_{yy}$. Dette er en enkel linær partielle differensialligning. Diskretisering av denne med en endelig differanse over et rektangulært grid gir følgende skjema

$$U_{i,j}^{n+1} = U_{i,j}^n + \frac{k}{h^2}\left(U_{i+1,j}^n + U_{i-1,j}^n + U_{i,j-1}^n + U_{i,j+1}^n - 4U_{i,j}^n\right)$$

Verdiene på et gitt tidssteg blir beregnet ved å kombinere fem verdier fra forrige tidsskritt, se Figur 3. Skjemaet kan derfor også gis en fortolkning som et standard glattingsfilter (Gauss-konvolusjon) i bildebehanding.



Figur 3. *Neste fragment bruker 5 verdier fra teksturen fra forrige tidskritt for løsning av varmeligningen med et forlengs differanseskjema.*

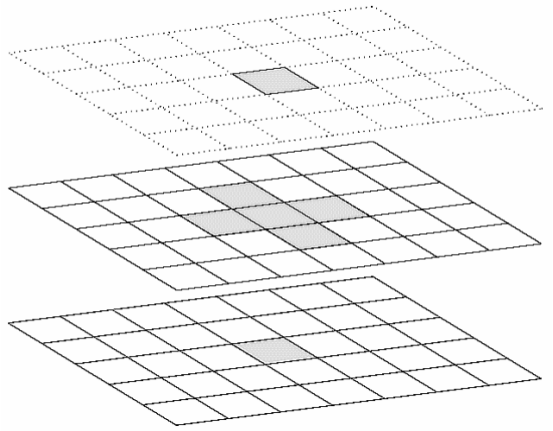
Det er enkelt å implementere skjemaet som en fragmentshader, se Listing 1. Den resulterende koden er svært lik en funksjon man kunne ha lagd i C (eller et lignende språk) for å beregne et steg av varmeligningen i et gridpunkt. De eneste forskjellene er i måten man henter ut dataene man skal prosessere – her ved hjelp av heatTex, som er en handle/peker til en 2D tekstur – og i måten man returnerer beregnede data på – ved å skrive dem ut som en teksturfarge i `gl_FragColor`).

Lineær bølgelikning

Den partielle differensiallikningen for lineære bølger er gitt ved $u_{tt} = u_{xx} + u_{yy}$. Dersom vi diskretiserer ligningen ved en standard differansemetode får vi et enkelt eksplisitt skjema:

$$U_{i,j}^{n+1} = 2U_{i,j}^n - U_{i,j}n - 1 + \frac{k^2}{h^2}\left(U_{i+1,j}^n + U_{i-1,j}^n + U_{i,j-1}^n + U_{i,j+1}^n - 4U_{i,j}^n\right)$$

Her blir overflatehevningen i et punkt ved tidssteg $(n+1)$ beregnet fra fem verdier fra n^{te} tidssteg og en verdi fra $(n-1)^{\text{ste}}$ tidssteg. Listing 2 viser hvordan skjemaet kan implementeres på en GPU.



Figur 4. *Fragment ved tidssteg $(n+1)$ bruker fem verdier fra teksturen for nte tidssteg og en verdi fra teksturen for $(n-1)^{ste}$ tidssteg når den lineære bølgelikning løses ved et eksplisitt differanse-skjema.*

Slik den står, inneholder ikke bølgeligningen noen dempningsledd. Dersom man ønsker å bruke denne ligningen for å generere fysiske effekter i en mer visuell setting, kan man f.eks legge inn et lineært dempningsledd proporsjonalt med den tidsderiverte av overflatehevningen.

Flodbølger

Figur 5 viser stillbilder fra en GPU simulering av et dambrudd i et fjellterreng. Dambruddet er beskrevet ved hjelp av de såkalte Saint-Venant ligningene, som består av et ikkelineært sett med bevarelseslover for masse og moment. Ligningene modellerer overflatebølger som blir dannet av gravitasjonskrefter og bunntopografi, men tar ikke hensyn til viktige faktorer som brunnfriksjon, turbulens, mm. Som diskretiseringsskjema har vi valgt en moderne høyoppløslighetsmetode som gir nøyaktig oppløsning av glatte bølger så vel som brytende bølger (i form av sjokk). Spesielt er metoden i stand til å simulere overgangen mellom vann og tørt land, samt såkalte (kvasi)stasjonære bølger. For denne type simuleringer, har vi observert en faktor 10 i ytelsesforbedring sammenlignet med tilsvarende CPU simuleringer. Simuleringen som er vist i figur 5 er utført på et regulært 256×256 grid, og her krever faktisk visualiseringen mer GPU kraft enn selve simuleringen av overflatebølgen.

GPU programmering – noe for deg?

I denne artikkelen har vi forsøkt å gi et innblikk i noe vi syns er et nytt og svært spennende fagfelt: "General-Purpose Computation Using Graphics Hardware". For mer informasjon om fagfeltet, kan man se følgende webside: `http://www.gpgpu.org`.

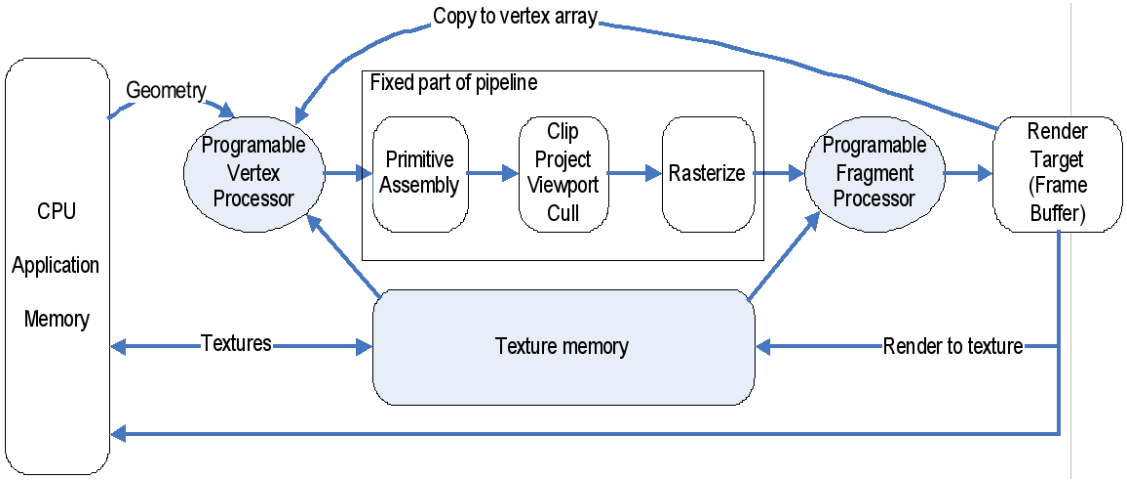
Vår konklusjon så langt er at GPU-teknologien har et stort potensial for å brukes til ikke-grafiske formål, både innenfor akademia og i industrien. I dag har en GPU en regneytelse som er omtrent 10 ganger større enn den teoretiske regneytelsen til en CPU, og denne forskjellen forventes å øke mye mer i årene fremover. Mens regnekraften i en CPU i en årrekke har fulgt Moores lov (med en dobling i ytelse hver 18.måned), har GPUene fulgt Moores lov kvadrert (dvs en firedobling i ytelse hver 18.måned). De som har fulgt med i diskusjonene rundt den nye Cell-prosessoren, vet også at det er andre spennende ting på gang, utover det å kun øke antall pipelines på GPUene. (Dette er kun en konstatering; vi ønsker ikke å ta standpunkt til hvorvidt Cell er en hype eller ikke; det gjenstår å se).

Slik vi ser det, er måten GPUer programmeres på den viktigste begrensningen for utstrakt bruk av GPUer som en aksellerator for generelle beregninger. Det å skrive fragmentshadere er i og for seg ikke så vanskelig, men hvis du har lite erfaring i grafikkprogrammering, og begrenset kunnskap om den grafiske pipeline, kan det være vanskelig å finne ut av hvordan man skal sette det hele opp for å få ulike shadere til å spille sammen i en (ikke-grafisk) applikasjon.

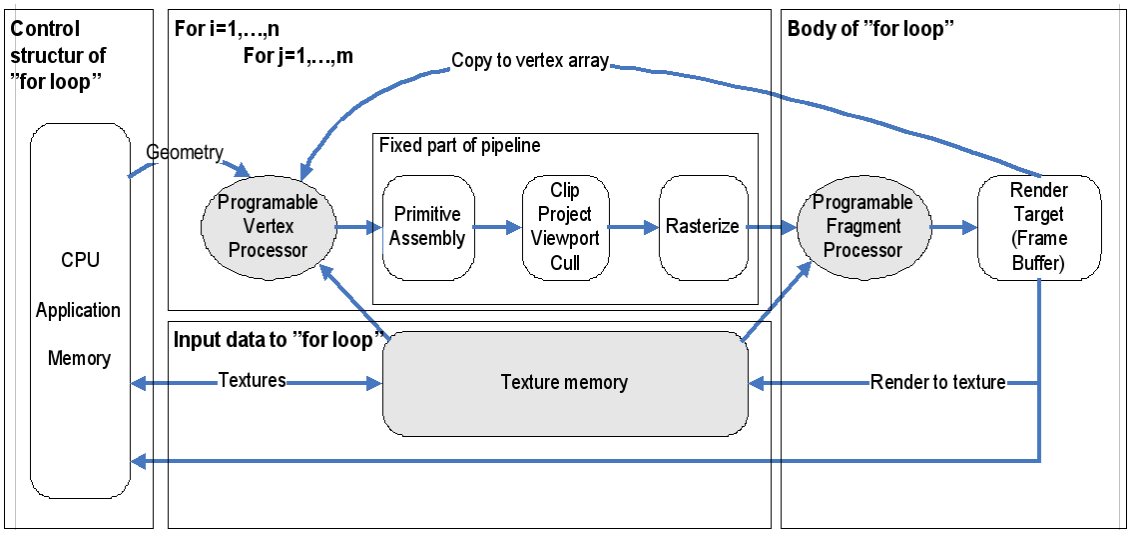
Vår erfaring er at det beste for nybegynnere er å starte fra eksisterende applikasjoner med shadere som virker. Disse kan så gradvis modifiseres til å gjenspeile den ønskede funksjonalitet. Når en oppdager at en shader ikke virker som ønsket, kan dette ha flere årsaker:

- Det kan være en feil i algoritmen.
- Du kan ha misforstått funksjonaliteten til shaderspråket.
- Det kan være feil i GPUens drivere!

Det finnes per i dag svært begrenset med debuggere for shadere, så debugging må i stor grad baseres på å lese verdier fra teksturer og analysere disse. For å forstå hva shadere gjør kan være nyttig å visualisere verdiene i bergningsnettet som et bilde. Når shadere brukes som en "løkkeakselerator", noe vi kommer tilbake til, kan det være nyttig å sammenlikne en CPU/GPU implementasjon med en ren CPU-implementasjon. Også for å måle for å måle ytelsesøkningen.



Figur 1. *En forenklet oversikt over den grafiske pipeline og programmerbare vertex- og fragmentprosessorer. Legg merket til at for å få fragmentprosessoren til å starte, må alle steg i prosessen før fragmentprosessoren være gjennomløpt. En geometri må derfor "eie" dataene for at de skal kunne benytte fragmentprosessoren som regneressurs.*



Figur 2. *For–løkke implementert i fragmentprosessoren.*

[Varmeligning Vertexshader]

```
varying vec4 texXcoord;
varying vec4 texYcoord;
uniform vec2 dXY;

void main(void)

  texXcoord = gl_MultiTexCoord0.yxxx
     + vec4(0.0,0.0,-1.0,1.0)*dXY.x;
  texYcoord = gl_MultiTexCoord0.xyyy
     + vec4(0.0,0.0,-1.0,1.0)*dXY.y;

  gl_Position = gl_ModelViewProjectionMatrix
     * gl_Vertex;
```

[Varmeligning Fragmentshader]

```
varying vec4 texXcoord;
varying vec4 texYcoord;

uniform sampler2D heatTex;
uniform float r;

void main(void)

  vec4 col;
  vec4 tex = texture2D(heatTex, texXcoord.yx);
  vec4 tex0 = texture2D(heatTex, texXcoord.zx);
  vec4 tex1 = texture2D(heatTex, texXcoord.wx);
  vec4 tex2 = texture2D(heatTex, texYcoord.xz);
  vec4 tex3 = texture2D(heatTex, texYcoord.xw);

  col = tex + r*(tex0+tex1-4.0*tex+tex2+tex3);
  gl_FragColor = vec4(col);
```

Listing 1. *Fragment- og vertexshader for varmeligning løst med forlengs differanseskjema.*

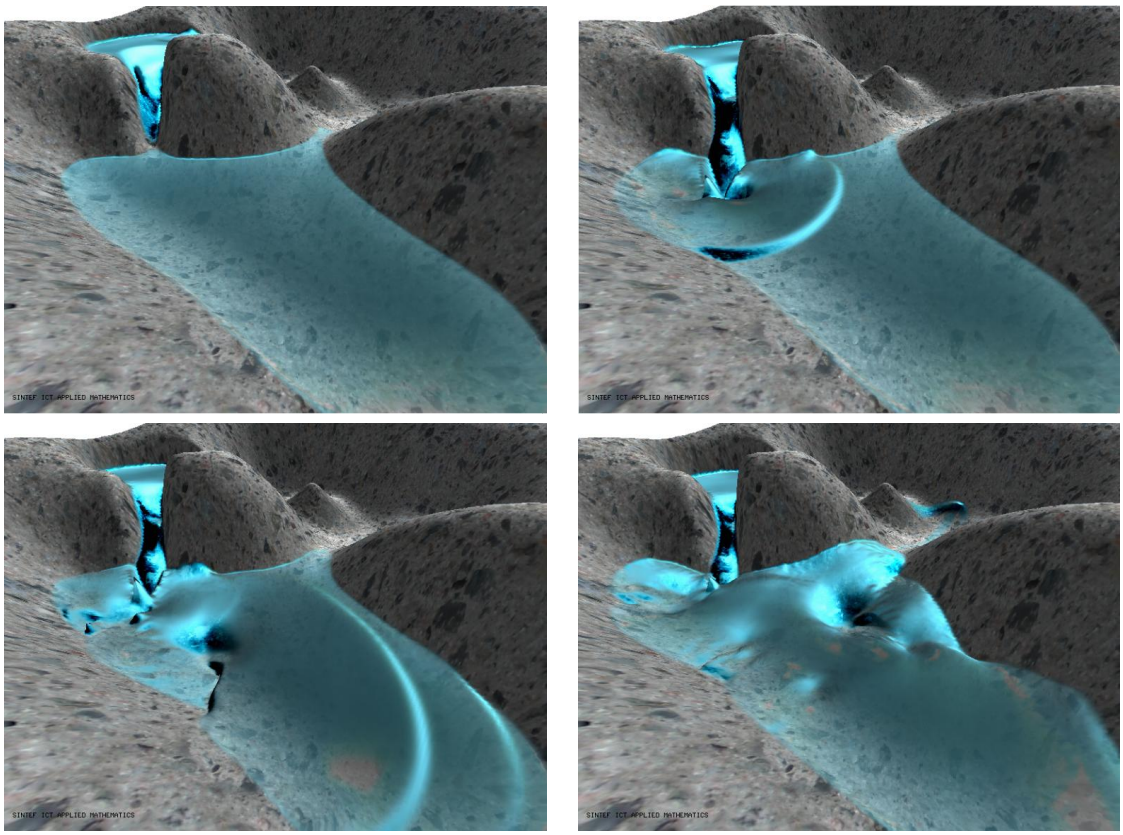
[Lineær bølgeligning Vertex shader]

```
varying vec4 texXcoord;
varying vec4 texYcoord;
uniform vec2 dXY;

void main(void)
texXcoord=gl_MultiTexCoord0.yxxx
    + vec4(0.0,0.0,-1.0,1.0)*dXY.x;
texYcoord=gl_MultiTexCoord0.xyyy
    + vec4(0.0,0.0,-1.0,1.0)*dXY.y;

gl_Position = gl_ModelViewProjectionMatrix
    * gl_Vertex;
```

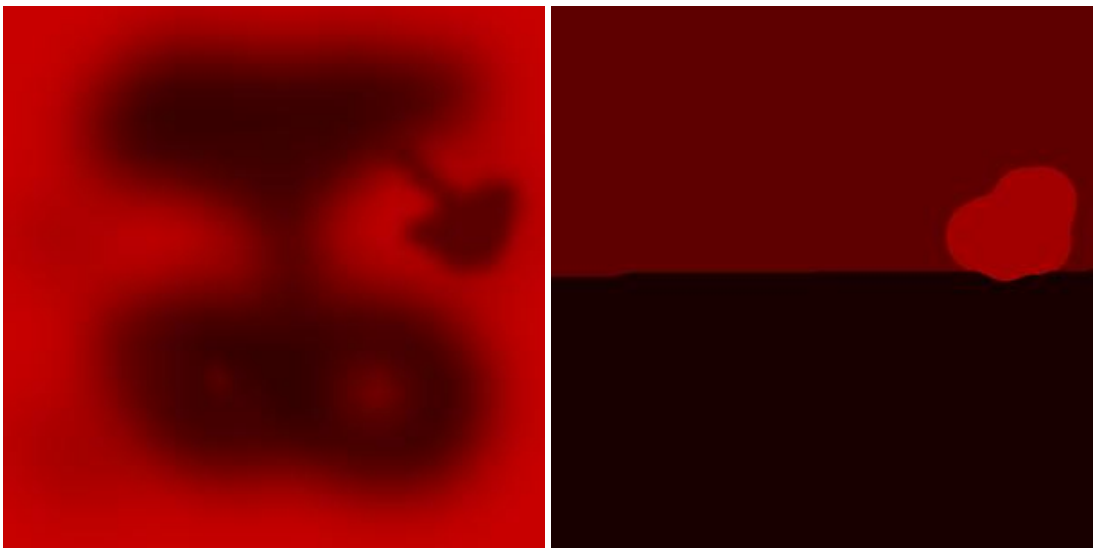
[Lineær bølgeligning Fragment shader]

```
varying vec4 texXcoord;
varying vec4 texYcoord;

uniform sampler2D texCurrent;
uniform sampler2D texLast;

void main(void)

  vec4 col;
  vec4 tex = texture2D(texCurrent, texXcoord.yx);
  vec4 tex0 = texture2D(texCurrent,texXcoord.wx);
  vec4 tex1 = texture2D(texCurrent,texXcoord.zx);
  vec4 tex2 = texture2D(texCurrent,texYcoord.xw);
  vec4 tex3 = texture2D(texCurrent,texYcoord.xz);
  vec4 texL = texture2D(texLast, texXcoord.yx);

  gl_FragColor = (2.0 * tex - texL + (0.5)*(tex0
      + tex1 + tex2 + tex3 - 4.0*tex))*0.92;
```

Listing 2. *Fragment- og vertexshader for løsning av den lineære bølgelikning med et sentrert differanse-skjema.*



Figur 5. *Visuell simulering av et dambrudd. Dambruddet skaper en flodbølge som flømmer ned en trang dal og inn i en liten innsjø hvor den gir opphav til tsunamibølger. Etterhvert som dalen fylles av vann, beveger flodbølgene seg ned i neste dal.*



Figur 6. *Bildet til venstre viser terrenghøyden i bølgesimuleringen som en fargelagt tekstur. Denne teksturen benyttes for å beskrive terrenget i simuleringen. Bildet til høyre viser vannhøyde for de forskjellige deler av en fargelagt tekstur. Denne teksturen benyttes for å beskrive vann nivå ved starten av simuleringen.*

Hva er NORSIGD?

NORSIGD – Norsk samarbeid innen grafisk databehandling – ble stiftet 10. januar 1974. NORSIGD er en ikke-kommersiell forening med formål *å fremme bruken av, øke interessen for, og øke kunnskapen om grafisk databehandling i Norge.*

Foreningen er åpen for alle enkeltpersoner, bedrifter og institusjoner som har interesse for grafisk datbehandling. NORSIGD har per januar 2004 25 institusjons-, 37 personlige og 5 EG-medlemmer. Medlemskontingenten er 1.000 kr per år for institusjoner. Institusjonsmedlemmene er stemmeberettiget på foreningens årsmøte, og kan derigjennom påvirke bruken av foreningens midler.

Personlig medlemskap koster 250 kr per år. Personlige medlemmer får tilsendt medlemsbladet *NORSIGD Info*. Kontingenten er redusert til 150 kr ved samtidig medlemskap i vår europeiske samarbeidsorganisasjon *Eurographics*.

Alle medlemmer får tilsendt medlemsbladet *NORSIGD Info* 2–4 ganger per år. NORSIGD har tilrettelagt informasjon om foreningen på Internett på adressen `http://www.norsigd.no`. Der finnes det også informasjon om GPGS, samt tidligere utgaver av *NORSIGD Info*.

Interesseområder

NORSIGD er et forum for alle som er opptatt av grafiske brukergrensesnitt og grafisk presentasjon, uavhengig av om basisen er *The X window System*, *Microsoft Windows* eller andre systemer. NORSIGD arrangerer møter og seminarer, formidler informasjon fra internasjonale fora og distribuerer fritt tilgjengelig programvare. I tillegg formidles kontakt mellom brukere og kommersielle programvareleverandører.

NORSIGD har lang tradisjon for å støtte opp om bruk av datagrafikk. Foreningen bidrar til spredning av informasjon ved å arrangere møter, seminarer og kurs for brukere og systemutviklere.

GPGS

GPGS er en 2D- og 3D grafisk subrutinepakke. GPGS er maskin- og utstyrsuavhengig. Det vil si at et program utviklet for et operativsystem med f.eks. bruk av plotter, kan flyttes til en annen maskin hvor plotteren er erstattet av en grafisk skjerm uten endringer i de grafiske rutinekallene. Det er definert grensesnitt for bruk av GPGS fra FORTRAN og C.

Det finnes versjoner av GPGS for en rekke forskjellige maskinplattformer, fra stormaskiner til Unix arbeidsstasjoner og PC. GPGS har drivere for over femti forskjellige typer utsyr (plottere, skjermer o.l.). GPGS støtter mange grafikkstandarder slik som Postscript, HPGL/2 og CGM. GPGS er fortsatt under utvikling og støtter stadig nye standarder.

GPGS eies av NORSIGD, og leies ut til foreningens medlemmer.

Eurographics

NORSIGD samarbeider med Eurographics. Personlige medlemmer i NORSIGD får 20 SFr rabatt på medlemskap i Eurographics, og vi formidler informasjon om aktuelle aktiviteter og arrangementer som avholdes i Eurographics-regi. Tilsvarende får Eurographics medlemmer kr 100 i rabatt på medlemskap i NORSIGD.

Eurographics ble grunnlagt i 1981 og har medlemmer over hele verden. Organisasjonen utgir et av verdens fremste fagtidsskrifter innen grafisk databehandling, *Computer Graphics Forum*. *Forum* sendes medlemmene annen hver måned. Eurographics konferansen arrangeres årlig med seminarer, utstilling, kurs og arbeidgrupper.

NORSIGD
v/ Reidar Rekdal
Postboks 290
1301 Sandvika

Returadresse:
NORSIGD v/ Reidar Rekdal
Postboks 290
1301 Sandvika

Styret i NORSIGD 2005

Funksjon	Adresse	Telefon	email
Leder	Ketil Aamnes DNV Software Veritasveien 1 1322 Høvik		Ketil.Aamnes @dnv.com
Fagansvarlig	Wolfgang Leister Norsk Regnesentral Postboks 114 Blindern 0314 OSLO	22 85 25 78 (direkte) 22 85 25 00 (sentralbord) 22 69 76 60 (fax)	leister@online.no
Sekretær	Reidar Rekdal Norsigd Postboks 290 1301 Sandvika	67 57 73 18 (direkte) 67 57 72 50 (sentralbord) 67 57 72 72 (fax)	reidar.rekdal @dnv.com
Styremedlem	Tor Dokken SINTEF IKT Postboks 124 Blindern 0314 OSLO	22 06 76 61 (direkte) 22 06 73 50 (fax)	Tor.Dokken @sintef.no
Varamedlem	Svein Taksdal Norges Vassdrags- og Energiselskap Hydrologisk Avdeling, Seksjon data Postboks 5091, Majorstua 0301 OSLO	22 95 92 86 (direkte) 22 95 92 01 (fax)	svein.taksdal @nve.no
Varamedlem	Magnar Granhaug ProxyCom AS Klæbuvn. 194 7037 Trondheim	73 95 25 00 97 72 76 98 (mobil) 73 95 25 09 (fax)	Magnar.Granhaug @proxycom.no

Svarkupong

- ◯ Innmelding – institusjonsmedlem (Kr 1000)
- ◯ Innmelding – personlig medlem (Kr 250)
- ◯ Innmelding – Eurographics medlem (Kr 150)
- ◯ Ny kontaktperson
- ◯ Adresseforandring

Navn: .....
Firma: .....
Gateadresse: .....
.....
Postadresse: .....
.....
Postnummer/sted: .....
.....
Telefon: .....
Telefaks: .....
email: .....