GPU-based Simulation of Cold Air Flow for Environmental Planning

Stephan Nowatschin[a], Martin Bertram[a] and Christoph Garth[a]

[a]TU Kaiserslautern

Simulating the effects of different soil types regarding flow resistance and cold air production is important for controlling air quality araound urban areas. In this paper we present a mathematical model and a simulation method for this problem. This model describes the cold air flow to be composed of two variables. The first is the velocity field which depends on flow resistance and the flow gradient. The second variable is a height field of the cold air which depends on cold air production and advection. To accelerate the simulation and its visualization, it is adapted to run on a GPU(Graphical Processing Unit). Implementing the simulation on fragment shaders makes it possible to render the height field of the landscape and a color map associated with the cold air height. In two passes we compute the cold air height for each time step and render the result to a texture. In a third pass, we render the height field of the landscape using this texure as color map.

1. Motivation

During stationary temperature inversions the air convection in urban areas, which lie close-by mountainous landscapes, is affected through cold air flow. Air quality in urban areas and settlements highly depends on this cold air flow. Sensitive changes in the flow intensity or direction may have serious consequences, such as smog and heat. Over night the different soils at the slopes cool down addicted to the property of the soil causing the air near by to cool down faster than the air on the same niveau over the valley. Due to the landscape's elevation gradient, the cold air starts to flow downwards creating a downwind. The cold air flow is merged at the settlement areas in the valleys and is known as katabatic wind. The characteristics of this cold air flow depends both on the height relief of the landscape and on the thermal and mechanical properties of the terrain. This means, that a different agricultural use of the soils may affect the cold air flow and with it the air convection in a settlement area. This can cause a sensitive change of air quality in these areas.

To consider the cold air flow during the planning of new sattlement areas or the changing of existing structures, it is necessary to simulate this cold air flow. Based on such a simulation, for example, we can calculate the effects of new structures, like streets, bridges and buildings at the city limit, which might cut the city from the cold air flow and thus degrade its air quality. Further more it is possible to simulate the impact of changes in the agricultural areas around a city on the cold air flow. These agricultural changes strongly affect the magnitude of the cold air flow, since, for example, an area which is used as acre cools down the air much faster than a forest. Moreover, the flow resistance of an acre is less than the resistance of a forest so that the velocity of the cold air flow is much higher over an acre. To support an environmental planning it is very helpful to have a fast simulation of such effects.

The simulation example presented here is a part of a valley at the Mosel, the Geisbachtal, where different agricultural soils exist (Figure 1) and therefore it is possible to show the influence of different soils on the production of cold air during a night.

In the next section, we summarize related work. Section 3 of this paper describes the continuous and discrete mathematical model of our simulation. Section 4 contains our simulation results for a selected landscape. Finally we conclude and discuss some extensions for future work in section 5.

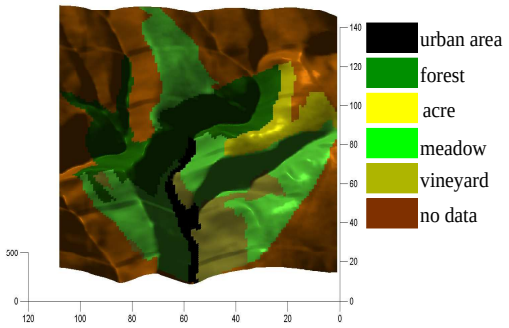

Figure 1. Soil types stimulating cold air production for our simulation

2. Related work

Stam [1] uses the Helmholtz-Hodge decomposition to obtain a stable solution for the Navier-Stokes equation. This is performed in a multi-level process combining extern forces, advection, diffusion and a divergence-free projection of the velocity field. This method can be used for the simulation of smoke [4]. Harris [3] implements the discretization of this multi-level process on the GPU. Each process step is implemented in a fragment program and each fragment program writes its results into a texture used by the next fragment program. We have not found any mathematical models defining cold-air flow in the literature. The phenomenon of cold-air flow has been described in [5][6], where only a vague definition of cold-air production and flow resistance is provided. Our contribution is a simple mathematical model describing this phenomenon and a GPU-based solver.

3. Algorithm

3.1. Mathematical Model

Before we derive a mathematical model for the described problem we define some quantities that are defined on a domain $D \subset \mathbb{R}^2$:

- Soil type : S={meadow,forest,vineyard, acre,settlement}
- Geographic height :$h = h(x, y)$
- Cold air height : $k = k(x, y, t)$
- Velocity : $\vec{v} = \vec{v}(x, y, t)$
- Agricultural usgae : $l = l(x, y)\epsilon S$
- Cold air production p : $S \rightarrow \mathbb{R}$
- Flow resistance w: $S \rightarrow \mathbb{R}$

For every time step, the cold air height k(x,y,t) and the associated velocity $\vec{v}(x, y, t)$ need to be computed for each point $(x, y)\epsilon D$. The related mass flow (flux) is calculated by:

$$\vec{F}(x, y, t) = k(x, y, t) * \vec{v}(x, y, t) \quad (1)$$

where * denotes the componentwise multiplication with a scalar[2].
The variation of the cold air height $\dot{k}(x, y, t)$ depends on a source term $p(x, y) := p(l(x, y))$, defining how much cold air is produced in a point (x, y) during a period of time, and on an advection term defining the transport of the cold air. In our model, diffusion has small impact and is neglected.
The advection term in a cell (x, y) depends on the velocity field and the gradient of the cold air height ∇k. The cold air flows contrary to this gradient and so the advection term results in $-\vec{v} \cdot \nabla k$, where $\cdot$ denotes the inner product. Hence, the change of the cold air height in a point (x, y) is:

$$\dot{k}(t) = p(x, y) - \vec{v}(x, y, t) \cdot \nabla k(x, y, t) \quad (2)$$

Further more equation (1) requires that the velocityfield is free of divergence to avoid sinks and sources of cold air in the advection term[2], i.e.:

$$div\ \vec{v} := \partial v_x/\partial x + \partial v_y/\partial y = 0.$$

To calculate the variation of the cold air height, the corresponding velocity $\vec{v}(x, y, t)$ is required. the velocity depends on the gradient of the slope, the cold air height and on a factor $1/w$. This factor depends on different agricultural soils and is a measure for the flow resistance $w(x, y) :=$

$w(l(x,y))$. In a simulation this parameter can be used to adjust the velocity.
These considerations provide the velocity in a point (x,y):

$$\vec{v} = -1/w(x,y) * (\nabla(h(x,y) + k(x,y,t))) \quad (3)$$

Our model does not consider inertia of air. Based on empirical data [6] we found that the velocity is rather linear in the gradient of the upper cold-air boundary $\nabla(h+k)$ and that the effect of acceleration can be neglected.. To calculate the cold air flow from equations (2) and (3), these have to be discretized.

3.2. Discretization

We now define the height field over a regular grid $P = \{0,...,m-1\} \times \{0,...,n-1\}$, assuming that the cold air is constantly distributed over each grid cell $P_{i,j}$. Hence, for each cell the cold air height in the center of the cell is computed and the resulting value is taken for the whole cell.
First, we compute the velocity of the flow across each cell boundary. Hereby, the velocity $\vec{v}$ is decomposed in its components v_x and v_y, defined at cell boundaries, see figure **??**. This way we know the velocity of the flow between each pair of adjacent grid cells. This approach is know as staggered grid, where vector quantities are represented at the boundaries between cells[3].
With equation (2) the velocity component in x-direction is calculated out of the height difference of the cells $P_{i,j}$ und $P_{i+1,j}$, multiplied with the factor $1/w(x,y)$.

$$v_{i,j\ ,x} = (h_{i,j} + k_{i,j} - h_{i+1,j} - k_{i+1,j}) * 1/w(x,y) \quad (4)$$

Hence, the component in y-direction is:

$$v_{i,j\ ,y} = (h_{i,j} + k_{i,j} - h_{i,j+1} - k_{i,j+1}) * 1/w(x,y) \quad (5)$$

Based on equations (3) and (4) the change of the cold air height is discretized. Here the change in a cell depends on the produced cold air $p(x,y)$ and on the in- and outflow. An outflow from the current cell to an adjacent cell occurs if the appropriate velocity for this transition is positive. Otherwise cold air flows from the neighbor to the current cell. This in- and outflow is calculated for each transition. We need to guarantee that not more cold air can flow out of a cell as really

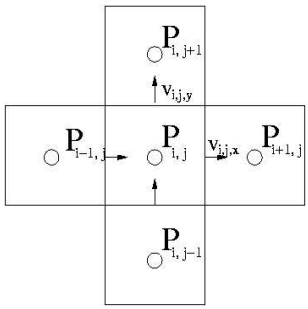

Figure 2. Grid cell with appropriate velocity vectors

available. With this assumption the flow out of a cell can be calculated as $v(x,y,t) * k(x,y,t)$. To avoid flow over multiple cell we constitute for the discretization that the sum of the outgoing velocities is not greater than one. Otherwise, we need to choose smaller time intervalls or we have to adapt the resistance for the agricultural soil.
Passing through the grid for each cell we need to calculate the flow to the right and the upper neighbor. Then the variation of the cold air height for $v_{i,j\ ,x} > 0$ during a time step dt is :

$$d_x k_{i,j}^{t+dt} = d_x k_{i,j}^t - v_{i,j,x} * k_{i,j}^t$$
$$d_x k_{i+1,j}^{t+dt} = d_x k_{i+1,j}^t + v_{i,j,x} * k_{i,j}^t$$

For $v_{i,j,x} < 0$ it is:

$$d_x k_{i,j}^{t+dt} = d_x k_{i,j}^t + v_{i,j,x} * k_{i+1,j}^t$$
$$d_x k_{i+1,j}^{t+dt} = d_x k_{i+1,j}^t - v_{i,j,x} * k_{i+1,j}^t$$

The variation of the height in the y-direction for $v_{i,j,y} > 0$ is:

$$d_y k_{i,j}^{t+dt} = d_y k_{i,j}^t - v_{i,j,y} * k_{i,j}^t$$
$$d_y k_{i,j+1}^{t+dt} = d_y k_{i,j+1}^t + v_{i,j,y} * k_{i,j}^t$$

For $v_{i,j,y} < 0$ it is :

$$d_y k_{i,j}^{t+dt} = d_y k_{i,j}^t + v_{i,j,y} * k_{i,j+1}^t$$
$$d_y k_{i,j+1}^{t+dt} = d_y k_{i,j+1}^t - v_{i,j,y} * k_{i,j+1}^t$$

After a complete iteration through the grid the new cold air height in an cell (i,j) results from:

$$k_{i,j}^{t+dt} = k_{i,j}^t + p(x,y) + d_x k_{i,j}^{t+dt} + d_y k_{i,j}^{t+dt} \quad (6)$$

This formulation assures, that the overal cold air concentration is conserved and in contrast to equation(2) it is not necassary that the velocityfield $\vec{v}$ is free of divergence. This has the advantage that no Helmholtz-Hodge decomposition based on an expensive Possion-equation is necessary, like, for example, in the Stable Fluids method[1]. The velocity components on the boundary of the grid can be set to a constant value which enables an outflow of the air out of the domain boundaries.

3.3. GPU-based Implementation

Our algorithm computes and renders the fast-motion cold air development in real time on a consumer-grade graphics card supporting Shader Model3. Prior to the rendering we propose two prerender steps. In each prerender step a rectangular region including our domain is drawn. Due to the dimension of these regions we can use each pixel as a grid cell and perform our calculations in a shader for each pixel.
Implementing the simulation in shaders brings up some differences compared to a CPU-based implementation. One difference is that array data are stored in textures. These textures can be read by the shader but cannot directly be manipulated by them. Normaly, the result of fragment shaders is written to the framebuffer and then displayed on the screen. To avoid this for our first two computation steps, we render the result of our calculations into a pbuffer. After rendering the entire region, we copy this pbuffer into a texture which is used in the next pass. The input of a shader is composed of different variables and textures and the output is a vector with four components which is normaly used as the color of the current pixel. This vector is the only way to get output from the shader.

We implement the discrete equation (6) in three steps. In the first step a fragment program is loaded into the fragment processor, which calculates the outflow of each cell to the four adjacent cells. This shader reads the cold-air heights for the current pixel and its four adjacent pixels, and it reads the flow resistance for the pixel. After loading this shader into the graphic processor, we create a pbuffer, in which the result of this shader is rendered. Beside the framebuffer, a pbuffer is an additional non-visible rendering buffer for an OpenGL renderer. Below, a part of our pbuffer implementation is shown:

```
Display    *display;
Display    *oldDisplay;
GLXPbuffer  pbuffer;
GLXPbuffer  oldDrawable;
GLXContext  context;
GLXContext  oldContext;

oldDisplay = glXGetCurrentDisplay();
oldDrawable = glXGetCurrentDrawable();
oldContext = glXGetCurrentContext();
int iScreen = DefaultScreen(oldDisplay);

GLXFBConfig *glxConfig;
int iConfigCount;

int pfAttribList[] =
{
  GLX_RED_SIZE,               32,
  GLX_GREEN_SIZE,             32,
  GLX_BLUE_SIZE,              32,
  GLX_ALPHA_SIZE,             32,
  GLX_FLOAT_COMPONENTS_NV,    true,
  GLX_DRAWABLE_TYPE,          GLX_PBUFFER_BIT,
  0,
};

glxConfig= glXChooseFBConfigSGIX(oldDisplay,
                                 iScreen,
                                 pfAttribList,
                                 &iConfigCount );
int pbAttribList[] =
{
  GLX_LARGEST_PBUFFER, true,
  GLX_PRESERVED_CONTENTS, true,
  0,
};

pbuffer= glXCreateGLXPbufferSGIX(oldDisplay,
                                 glxConfig[0],
                                 width,
                                 height,
                                 pbAttribList );
```

```
context=glXCreateContextWithConfigSGIX(
                    oldDisplay,
                    glxConfig[0],
                    GLX_RGBA_TYPE,
                    oldContext,
                    true );
}
```

After creating the pbuffer we set the pbuffer context as the current context, and then our rendering results are written to that pbuffer.

```
void makeCurrent()
{
  glXMakeCurrent(display,
                 pbuffer,
                 context);
}
```

Now we draw a rectangular region with OpenGL. After an ortographical projection and the viewport transformation this rectangle has the same size like our domain, so we can use each pixel in our fragment shader as a cell in our grid. The results from the pbuffer are copied back into a texture with glCopyTexSubImage2D and finally the old context is restored.

```
void restoreOld()
{
  glXMakeCurrent(oldDisplay,
                 oldDrawable,
                 oldContext);
}
```

We use the NV_texture_rectangle extension to provide float values not to be restricted on [0;1]. The second step in our implementation uses the ouput texture of first step to calculate the new cold-air height for each cell. To do this we load an other fragment shader into the fragment processor and repeat the instructions from step one. Finally we render the geographic height field using the results from step two in a third shader to compute the associated color map for the current time.

4. Results

We perform our simulation to calculate the cold air flow around the town Manubach, which is located in the Geisbachtal near the Mosel, see figure 3. The area around this town has different soil types like vineyard, acre, meadow and forest, see figure 1. The magnitudes we use for the cold air production and the flow resistance are listed in tables 1 and 2. In figure 5 - 11 the cold-air height is shown for every full hour. It can be observed that the cold air is merged in the valley and retained at the urban area, where it cannot flow as fast as on a free landscape. Furthermore it can be seen that the cold air from the meadows in the north of the area is blocked by the forest between the meadows and the settlement. This effect is also visible in the small velocities in this area (figure 4). The greatest cold air height is obtained in the eastern valley where on one side the flow is retained due to the settlement and on the other side of this area the cold air production is very high due to meadows and acres.
We also implemented our model in a CPU-base program and visualized the results with matlab. Our GPU-based program uses a pbuffer to render into in two steps.Hence, it is necassary to switch the render context before and after the prerender steps. This exchanging slows down our GPU-based implementation, but the advantage of the pbuffer is the possibility of using floating point

soil type	cold air production
vineyard	5 $[m^3/\text{h}]$
settlement	0 $[m^3/\text{h}]$
meadow	10 $[m^3/h]$
acre	12 $[m^3/\text{h}]$
forest	1 $[m^3/\text{h}]$

Table 1
Cold-air production per m^2.

soil type	resistance
vineyard	30 [%]
settlement	90 [%]
settlement	10 [%]
acre	5 [%]
forest	80 [%]

Table 2
Flow resistance in percent.

values grater than one. We obtained a computation time for our GPU-based implementation of about 500 msec for simulating the flow during one minute and rendering the result. This is about twice as fast as a software implementation. A greater speedup can be obtained, when the resolution of the data set is finer than in our example, where the grid was composed of 108×130 cells.

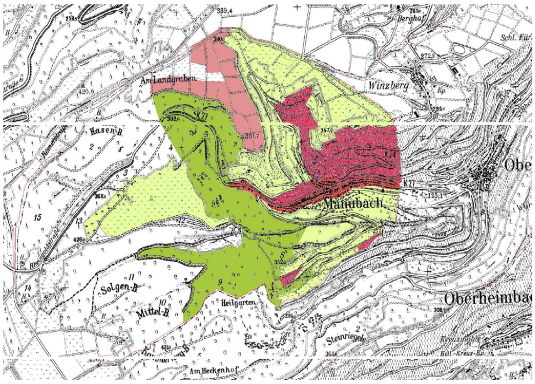

Figure 3. Map of the region

5. Conclusion

With our algorithm we can simulate the effects of different soil types regarding flow resistance and cold air production. This is important for controlling air quality around urban areas. In this paper we present a mathematical model based on advection describing the cold-air flow based on two variables. The first is the velocity field in which parameters like flow resistance and the flow gradient are introduced. The second variable is a height field of the cold air depending on cold-air productions and advection. With our model it is possible to show the correlation between these parameters and the effects of changing agricultural use. In a future work it would be interesting to find a more accurate mathematical description

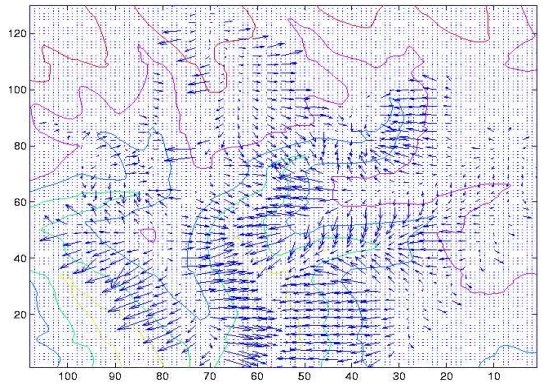

Figure 4. Velocities after 7h (created with matlab)

for the flow resistance and for the varying cold air production. A comparison of our simulation results with measurements is desired, to determine the exact resistance and production coefficients of a landscape.

6. Acknowledgements

We thank Gerd Reis, Gerik Scheuermann, and Robert Beckmann for their helpful support and discussion.

REFERENCES

[1] J. Stam : Stable Fluids, ACM Siggraph 1999, pp.121-128

[2] P.K. Kundu: Fluid Mechanics, Academic Press, 1990

[3] M.J. Harris: Fast Fluid Dynamics Simulation on the GPU, in GPU Gems, chapter 38, pp.637-665, 2004

[4] R. Fedkiw: Visual Simulation of Smoke, ACM Siggraph 2001

[5] A. Helbig, J.Baumüller, M.J. Kerschgens: Stadtklima und Luftreinhaltung, Springer Verlag, 1999 (In German)

[6] L. Finke: Regionale Luftaustauschprozesse und ihre Bedeutung für die räumliche Planung, in: Schriftenreihe 'Raumordnung' des BMBau 06.032, 1979 (In German)

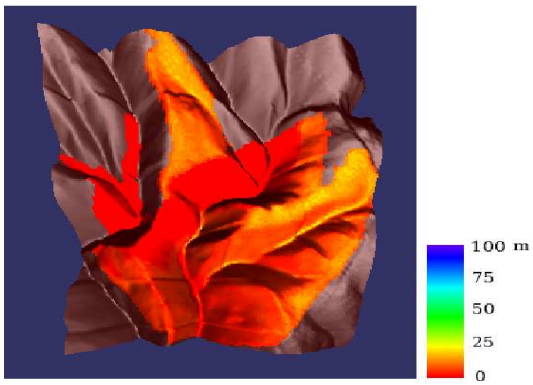

Figure 5. Cold air height after 1h

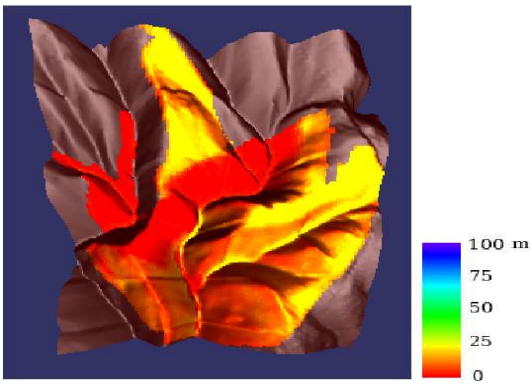

Figure 6. Cold air height after 2h

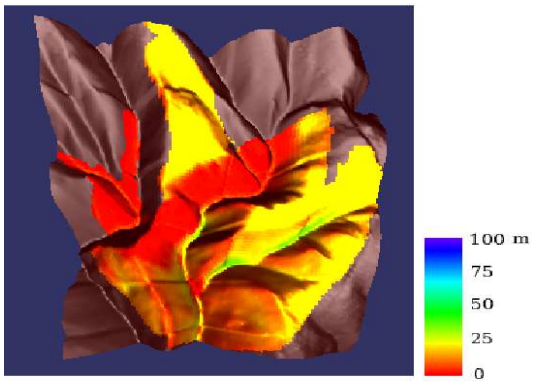

Figure 7. Cold air height after 3h

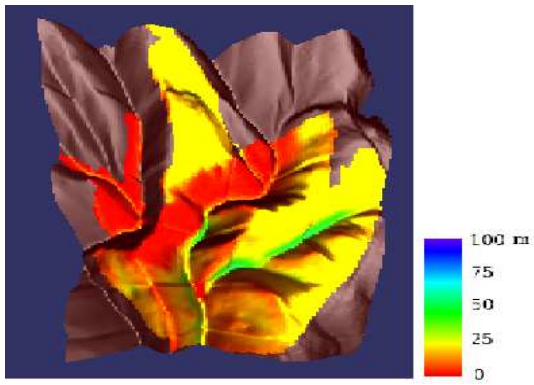

Figure 8. Cold air height after 4h

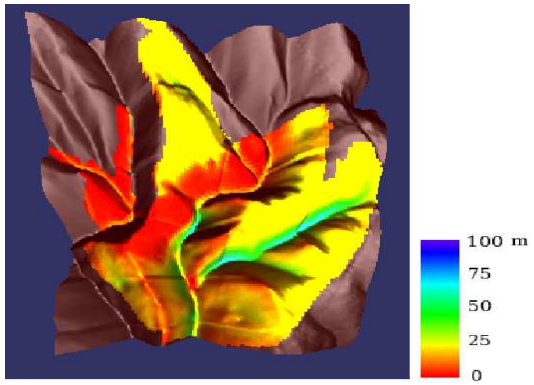

Figure 9. Cold air height after 5h

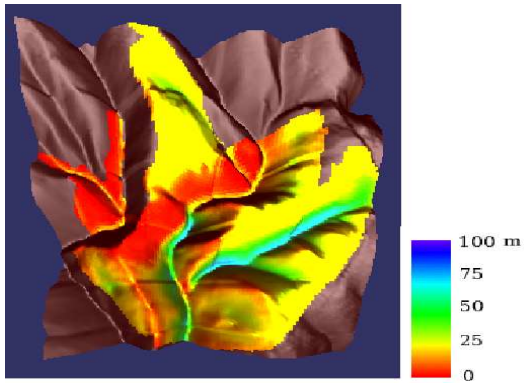

Figure 10. Cold air height after 6h

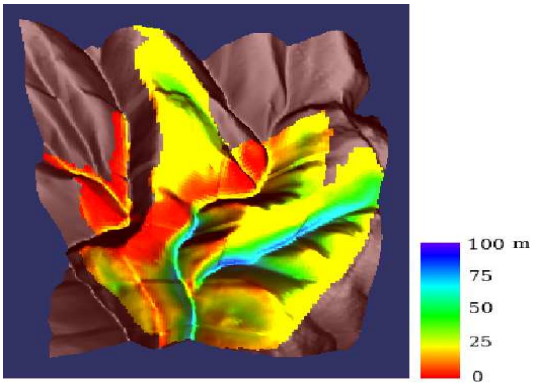

Figure 11. Cold air height after 7h