PROCEEDINGS, Fire and Evacuation Modeling Technical Conference (FEMTC) 2018
Gaithersburg, Maryland, October 1-3, 2018

SIMULATING REAL-TIME FIRE FOR FIREFIGHTING TRAINING

Dipl.-Inf. Christian Niemand, Prof. Dr.-Ing. M.Eng. Dieter Wloka

University of Kassel
Wilhelmshöher Allee 71-73
34121 Kassel, Hessen, Germany
e-mail: c.niemand@uni-kassel.de

ABSTRACT

In this paper we start with a short introduction why a real-time fire simulation is needed by showing a few use cases and the respective requirements. We continue with a short overview of related work and state of the art techniques.
Then we present our CUDA based parallel approach to simulate heat transfer, pyrolysis, transport and combustion in real-time followed by showing the results of our algorithms.
We will point out which components are still missing to reproduce the "Couch burning experiment" performed by Cape Girardeau Fire Department and to simulate a complete room fire in the future.

INTRODUCTION

In Germany firefighters are evaluating how 3D simulations could be used to improve education and tactical training. In this article we take a look on this area where a realistic fire and smoke behavior is necessary. Depending on the use cases the visualization and simulation requirements are different. We've created a list of configuration levels, which are mapped to special use cases. Currently we are working on level 3. The goal of this development is a minimal room fire simulation which is running in real-time with a minimum of 30 frames per second.

Table 1: Configuration levels of requirements and possible use cases

Level	Requirement	Use cases
1	Visualization of smoke Smoke spreading and smoke layer Fluid dynamics Voxelization of 3D geometry Thermal imaging	Search and rescue tactics Breathing apparatus training Smoke extraction from buildings Ventilation
2	Visualization of non-spreading fire	Leadership training (group level)
3	Visualization of spreadingfire Visualization of fire phenomena Visualization of decomposing objects Simple fuel based pyrolysis and combustion Heat transfer Extinguishing	Extinguishing techniques (cooling down room and smoke)
4	Reactions with different fuel types Complex pyrolysis and combustion	Measuring technique (Explosimeter)
5	Realistic parameter settings	-

RELATED WORK

We divide the field of fire simulation and visualization into three sections; Simulation applications, none real-time rendering software and real-time applications. Simulation applications like Fire Dynamics Simulator (FDS) have an excellent mathematical model [3] to compute the most realistic data. The calculations are very complex and taking too much time to consider this solution for real-time purposes. None real-time rendering software like Blender or Maya have a very high visual quality. They are using a reduced mathematical model but rendering time is still too high. Both sections are using computational fluid dynamics (CFD) to simulate fire and smoke.
The third and most relevant section for us are real-time applications. Commonly real-time applications are using simple particle systems to visualize fire and smoke. But this very fast technique is not suitable to simulate a realistic fire behavior. It is more advisable to use fluid dynamics to mimic a more realistic behavior of gases.
An advanced demo using real-time fluid dynamics for fire and smoke is shown by NVIDIA demonstration NvFlow [10]. It is highly optimized to show high quality fire in video games. The focus is on the visual aspect but not on real fire phenomena which are relevant for firefighter training. Melek [4] presents an approach including a burning process, chemical combustion, heat distribution, decomposition and deformation of burning solids.
Some video game simulations like Firefighting Simulator 2018 [9] are also using fluid dynamics to simulate fire. The foundations of using fluid dynamics in real-time are described in Stable Fluids [5] and Real-Time Fluid Dynamics for Games [6] by Jos Stam, in Fast Fluid Dynamics Simulation on the GPU [2] by Mark Harris and in Real-Time Simulation and Rendering of 3D Fluids [1] Keenan Crane, Ignacio Llamas and Sarah Tariq. An interesting fuel based idea to burn object surfaces is described in Voxels on Fire [7] by Ye Zhao et al.
An application which combines these techniques and supports a realistic real-time fire which is acceptable for firefighting training does not exist.

IMPLEMENTATION

To visualize our simulation we are using the game engine Unity 3D from Unity Technologies. Note that the visualization can be implemented in any other graphics engine as well. The simulation itself is using Nvidia CUDA and is implemented as a Windows dynamic link library which is used as a plugin for Unity 3D.

Initialization

The initialization is performed sequentially by the game engine shown in Listing 1. To voxelize the geometry we are using the mattatz voxelizer from Masatatsu Nakamura [8].

```
Create cubic simulation volume in game engine (256^3 voxels, (5.12m)^3);
Create all CUDA textures for simulation (see Table 2);
Create RGBA textures for rendering;
Voxelize scene geometry;
Copy voxel data (temperature, fuel type, etc.) into CUDA textures;
```

Listing 1: Initialization procedure in pseudocode

Simulation Loop

The simulation loop shown in Listing 2 is called every frame and is executed in parallel on the GPU. Because the simulation volume is a 3D grid, a 3D texture with the same resolution is an ideal data storage for data like velocity, temperature and fuel type etc. In our case we are working with a 256^3 grid which represents a $(5.12m)^3$ cube in the virtual scenario. For each cell the algorithm is executed in a separated thread on the GPU. To avoid race conditions ideally the algorithm only reads from other cells and writes to its own cell.

```
while simulation is running do
    Update game engine runtime data in CUDA plugin (e.g. debug switches);
    Transport vector and scalar fields;
    Perform conduction;
    Perform heat transfer;
    Perform radiation;
    Perform pyrolysis;
    Perform combustion;
    Add buoyancy to velocity field;
    Check boundary velocity (no slip);
    Calculate new velocity field;
    Convert simulation data into RGBA render textures;
end
```

Listing 2: Simulation loop in pseudocode

Table 2: Used textures

Name	Usage
Temperature air	Contains the air temperature (all gases).
Temperature solid	Contains the temperature in solid cells.
Heat sources	Heat sources are overwriting temperature cells every frame.
Oxygen	Contains the amount of oxygen in air cells.
Fuel solid	Contains the amount of solid fuel in solid cells.
Fuel gas	Contains the amount of gaseous fuel in air cells.

Fuel type	Defines the behavior of cells.
	`EMPTY: Cell contains gases and no solid fuel`
	`IGNORE: Cell is solid but ignores conduction and pyrolysis`
	`INCOMBUSTIBLE: Cell is solid but ignores pyrolysis`
	`PAPER: Cell is solid and uses attributes for paper`
	`WOOD: Cell is solid and uses attributes for wood`
	`METAL: Cell is solid and uses attributes for metal`
	`HEAT_SOURCE: Immutable heat source with fixed temperature`

Light	Contains the light emission produced by combustion.
Smoke	Contains the amount of smoke produced by combustion.
Velocity	Contains the velocity field which is used for transport.
Divergence	Necessary for calculating the next velocity field.
Pressure	Necessary for calculating the next velocity field.
Render Texture	Used by game engine to render all data.

Transport

The transport algorithm shown in Listing 3 is based on Stam, Harris and Crane and will not be explained in detail in this paper. Because texture sampling would steal data from solid cells temperature must be separated into gaseous and solid temperature textures.

```
if Thread cell is not solid and not a border cell then
    Read thread cell velocity;
    Trace velocity back depending on dt;
    Sample new vector (velocity) or scalar (temperature, smoke, etc.) from
      respective texture;
    if quantity supports fading (e.g. light emission) then
        Apply fading;
    end
    Write sampled and faded value into thread cell;
end
```

Listing 3: Transport algorithm based on Stam, Harris and Crane in pseudocode

Conduction

The algorithm in Listing 4 shows the calculation of conduction and is based on the conductibility of temperature equation (Eq. 1) which describes the thermal diffusivity in homogeneous and isotropic materials i.e. material attributes are the same in every voxel and it has no directions in conductivity.

```
if Thread cell is not a border cell then
    if Thread cell supports conduction then
        if Thread cell is heat source then
            Override thread cell temperature T_tc with heat source value T_hs;
        else
            Read thread cell temperature T_tc;
            foreach neighbor cell do
                if neighbor cell is solid then
                    Read neighbor cell temperature T_nc;
                else
                    Read thread cell temperature T_tc;
                end
            end
            Calculate new cell temperature T_tc^{N+1};    // Eq. 2, Eq. 3
        end
    end
end
```

Listing 4: Conduction algorithm in pseudocode

Heat Transfer

In our simulation heat transfer describes the process when temperature is exchanged between cells at solid and gas borders and vice versa. Currently only the exchange direction from solid to gaseous cells (shown in Listing 5) is implemented. This heat transfer is responsible to heat up gaseous cells which results in buoyancy.

```
if Thread cell is not a border cell then
    if Thread cell supports heat transfer then
        Read thread cell temperature T_tc;
        foreach neighbor cell do
            if neighbor cell is not solid then
                if Thread cell temp. T_tc > neighbor cell temp. T_nc then
                    Calculate lost temperature T_lost;                  // Eq. 4
                    Transfer temperature T_lost to neighbor cell;       // Eq. 5
                    Accumulate lost temperature T_lost to T_trans;      // Eq. 6
                end
            end
        end
        Subtract accumulated temp. T_trans from thread cell;           // Eq. 7
    end
end
```

Listing 5: Heat transfer algorithm from solid to gas direction in pseudocode

Pyrolysis

Depending on the cell temperature solid fuel is converted into gaseous fuel shown in Listing 6. Currently only cells with none solid neighbors are producing gaseous fuel. We are planning an advanced version of this algorithm where inner cells will also produce gaseous fuel, which will be transported and injected to the nearest non-solid cell.

```
if Thread cell is not a border cell then
    if Thread cell supports pyrolysis then
        Read thread cell temperature $T_{tc}$;
        Count neighbor cells which are not solid $N_{enc}$;
        Calculate released solid fuel $Fuel_{rel}$;                     // Eq. 8
        Subtract released solid fuel $Fuel_{rel}$ from thread cell;
        foreach neighbor cell do
            if neighbor cell is not solid then
                Convert rel. fuel $Fuel_{rel}$ to gas fuel $Fuel_{gas}$;       // Eq. 9
                Inject gaseous fuel $Fuel_{gas}$ into neighbor cell;    // Eq. 9
            end
        end
    end
end
```

Listing 6: Pyrolysis algorithm in pseudocode

Combustion

If a cell has an oxygen and gaseous fuel concentration which is within a reactive explosion range (Def. 1) a combustion happens, if the temperature is above an ignition temperature or the cell is touched by a flame. Depending on fuel and oxygen concentration the combustion varies in strength.

```
if Thread cell is not a border cell and cell is not solid then
    if oxygen and gaseous fuel are in explosion range then       // Def. 1
        if cell temp. > ignition temp. or cell has contact with flame then
            Calculate amount of reacting gas fuel $Fuel_{react}$;       // Eq. 10
            Calculate amount of reacting oxygen $Oxygen_{react}$; // Eq. 11
            Subtract reacting gaseous fuel $Fuel_{react}$ from thread cell;
            Subtract reacting oxygen $Oxygen_{react}$ from thread cell;
            Calculate combustion value $Comb$;                    // Eq. 12
            Calculate light emission $Light$;                     // Eq. 13
            Calculate smoke intensity $Smoke$;                    // Eq. 14
            Calculate radiation $Rad$;                            // Eq. 15
            Calculate new thread cell temperature $T_{tc}^{N+1}$;       // Eq. 16
        end
    end
end
```

Listing 7: Combustion algorithm in pseudocode

Buoyancy

Depending on temperature of adjacent cells buoyancy is added to the velocities y (up) component.

```
if Thread cell is not a border cell then
    if Thread cell is not solid then
        Read thread cell temperature $T_{tc}$;
        foreach neighbor cell do
            Read neighbor cell temperature $T_{nc}$;
        end
        Calculate ambient temperature $T_{amb}$;                       // Eq. 17
        Calculate turbulent buoyancy $Buo_{turb}$;                     // Eq. 18
        Calculate laminar buoyancy $Buo_{lam}$;                        // Eq. 19
        Calculate mixed buoyancy $Buo_{mix}$;                          // Eq. 20
        Add $Buo_{mix}$ to $y$ element of thread cell velocity $\vec{v}$;    // Eq. 21
    end
end
```

Listing 8: Buoyancy algorithm in pseudocode

Table 3: Equations

Conduction	$$\frac{\partial T(\vec{x},t)}{\partial t}=a\cdot\Delta(\vec{x},t)$$	(Eq. 1)
	$$T_{delta}=\left(\frac{T_{i+1,j,k}-2T_{i,j,k}+T_{i-1,j,k}}{(\delta x)^2}+\frac{T_{i,j+1,k}-2T_{i,j,k}+T_{i,j+1,k}}{(\delta y)^2}+\frac{T_{i,j,k+1}-2T_{i,j,k}+T_{i,j,k+1}}{(\delta z)^2}\right)$$	(Eq. 2)
	$$T_{tc}^{N+1}=T_{tc}^{N}+T_{delta}$$	(Eq. 3)
Heat Transfer	$$T_{lost}=(T_{tc}-T_{nc})k\cdot\delta t$$	(Eq. 4)
	$$T_{nc}^{N+1}=T_{nc}^{N}+T_{lost}$$	(Eq. 5)
	$$T_{trans}=\sum T_{lost}$$	(Eq. 6)
	$$T_{tc}^{N+1}=T_{tc}^{N}-T_{trans}$$	(Eq. 7)
Pyrolysis	$$Fuel_{rel}=T_{tc}\cdot p_{rr}\cdot\delta t$$	(Eq. 8)
	$$Fuel_{gas}^{N+1}=Fuel_{gas}^{N}+\frac{Fuel_{rel}}{N_{enc}}\cdot p_{cr}$$	(Eq. 9)
Combustion	$$Fuel_{react}=c_{fc}\cdot Fuel_{tc}\cdot\delta t$$	(Eq. 10)
	$$Oxygen_{react}=c_{oc}\cdot Oxygen_{tc}\cdot\delta t$$	(Eq. 11)
	$$Comb=c_c\cdot(Fuel_{react}+Oxygen_{react})$$	(Eq. 12)
	$$Light=c_{light}\cdot Comb$$	(Eq. 13)
	$$Smoke=c_{smoke}\cdot Comb$$	(Eq. 14)
	$$Rad=c_{rad}\cdot Comb$$	(Eq. 15)
	$$T^{N+1}=T^{N}+c_t\cdot Comb$$	(Eq. 16)
Buoyancy	$$T_{amb}=\frac{T_{i-1,j,k}+T_{i+1,j,k}+T_{i,j-1,k}+T_{i,j+1,k}+T_{i,j,k-1}+T_{i,j,k+1}}{6}$$	(Eq. 17)
	$$Buo_{turb}=(T_{tc}-T_{amb})\cdot b_{turb}\cdot\delta t$$	(Eq. 18)
	$$Buo_{lam}=T_{tc}\cdot b_{lam}\cdot\delta t$$	(Eq. 19)
	$$Buo_{mix}=Buo_{turb}\cdot(1-\alpha)+Buo_{lam}\cdot\alpha$$	(Eq. 20)
	$$v_y^{N+1}=v_y^{N}+Buo_{mix}$$	(Eq. 21)
	$$ExplRange=\{(x,y)\in Oxygen\times Fuel \mid Oxygen_{min}\leq x\leq Oxygen_{max}, Fuel_{min}\leq y\leq Fuel_{max}\}$$	(Def. 1)

RESULTS

This section shows the results of our real-time simulation using the algorithms presented above.

Voxelization

Figure 1 shows a 3D geometry of a wooden couch frame, which is converted into voxels shown in Figure 2. Every voxel can contain data like temperature and/or material attributes. The voxel data will be transferred into the according cell of the CUDA 3D textures.

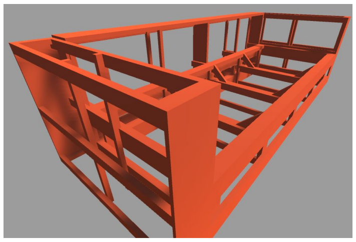

Figure 1: 3D geometry of a couch frame

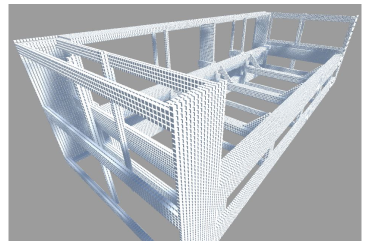

Figure 2: Voxelized couch frame

Conduction

Figure 3 shows the expected spreading of temperature in a test object. In this case a very high conductivity is used to cause a fast temperature propagation.

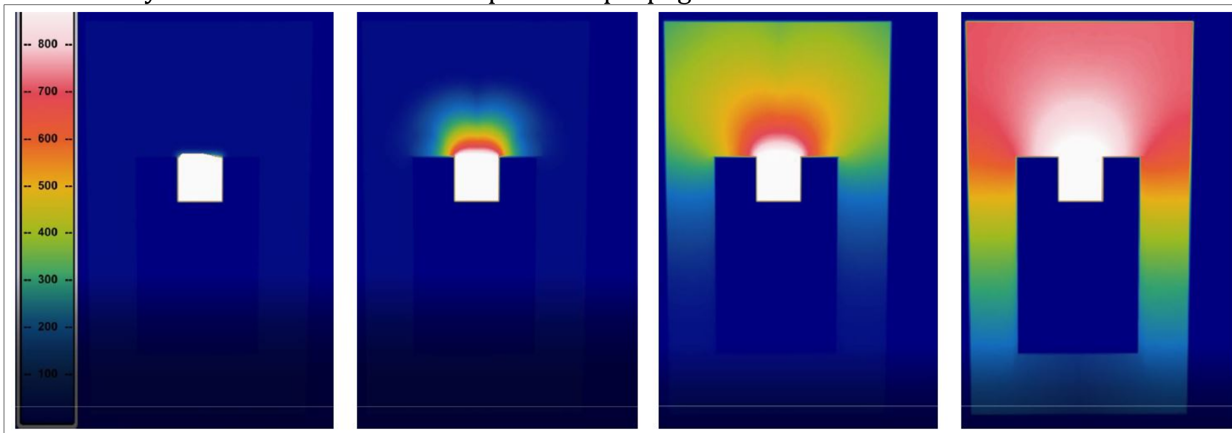

Figure 3: Temperature flow within a test object at time frames t=0.45s, t=1.74s, t=5.58s and t=13.22s

Heat Transfer And Buoyancy

Heat transfer between solid and gaseous cells is one reason how gaseous cells can increase temperature. Different temperatures in adjacent cells results in buoyancy. Figure 4 shows laminar buoyancy, Figure 6 turbulent buoyancy and Figure 5 mixed buoyancy.

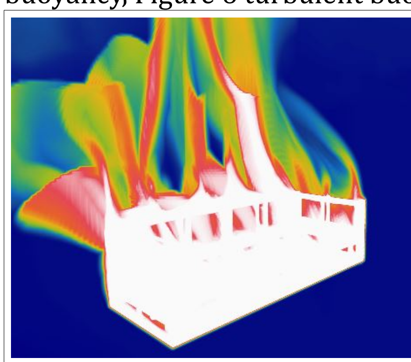

Figure 4: Laminar $\alpha = 1$

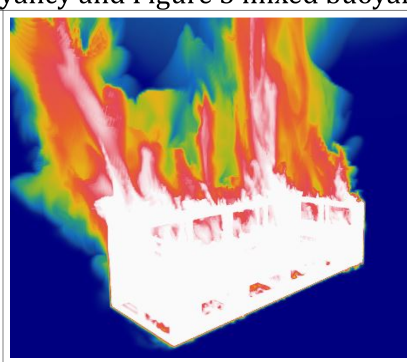

Figure 5: Mixed $\alpha = 0.5$

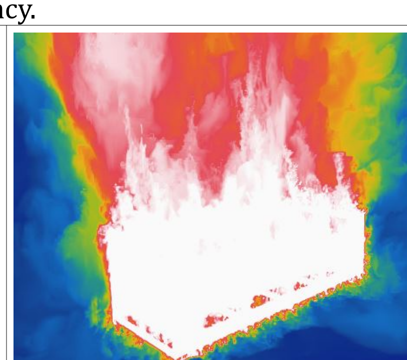

Figure 6: Turbulent $\alpha = 0$

Pyrolisis And Combustion

Figure 7 shows escaping gaseous fuel from a hot couch frame with enabled heat transfer from solid to gas and buoyancy. The transferred temperature results in buoyancy which diffuses the gaseous fuel. Figure 8 shows the same scenario but with disabled heat transfer and buoyancy. This results in no buoyancy and the gaseous fuel is gathering around the couch frame. Figure 9 shows the combustion process. The brighter the color the more intense is the combustion. Combustion also increases gas temperature in a cell and causes buoyancy.

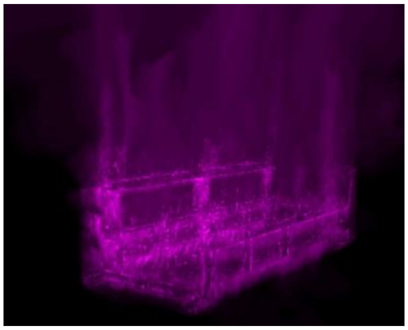

Figure 7: Pyrolysis with heat transfer and buoyancy

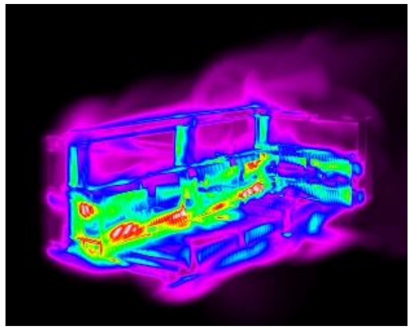

Figure 8: Pyrolysis without heat transfer and buoyancy

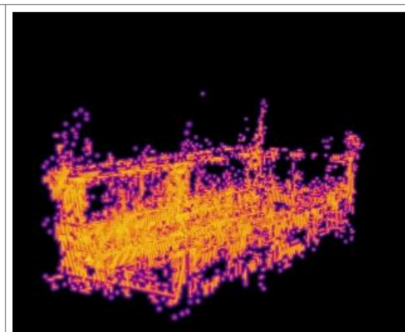

Figure 9: Combustion intensity

Burning Couch Frame

We have built a scenario with one couch frame and one heat source to start reconstructing the "Couch burning experiment" performed by Cape Girardeau Fire Department. The simulation runs with 35 frames per second on a single NVIDIA TitanXp and with 32 frames per seconds on a NVIDIA GeForce 780Ti. The simulation volume is a 256^3 grid. At the back right corner of the couch frame the ignition source is placed. Fire progresses from right to left side and combustion stops when no solid fuel is left and no gaseous fuel can be produced anymore.

Figure 8: Burning couch frame at time frames t=5s, t=60s, t=210s and t=290s

Stress Test

We also built a stress test to test the possibilities for more complex scenarios. We used a scenario with 18 stacked couch frames and 4 heat sources (Figure 9 and Figure 10). Figure 11 shows conduction within the stack of couch frames and Figure 12 shows the speed (no direction) of the velocity field. The brighter the color, the faster the velocity. The simulation runs with 35 frames per second on a Nvidia TitanXp (Intel i9 2.9Ghz, 64 GB RAM) and with 32 frames per seconds on a Nvidia GeForce 780Ti (Intel i7 3.6 Ghz, 16 GB RAM). The simulation volume and the performance at runtime are the same as they are in the single couch scenario but the voxelization process during initializing needs 18 times longer (approx. 90 seconds). Referring to benchmark data and videocard specifications the TitanXp should provide around 50 frames per seconds. The small difference between the measured frames per second reveals a bottleneck probably in our sequential implementation or a synchronization issue.

Figure 9: Stress test configuration

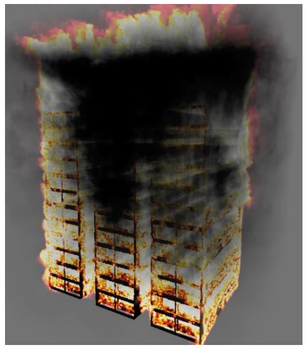

Figure 10: Stress test stacked burning couch frames

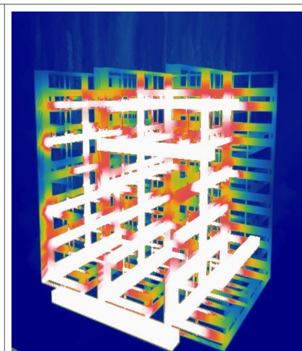

Figure 11: Stress test conduction

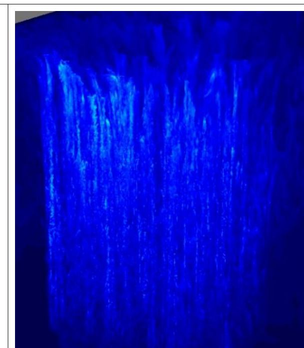

Figure 12: Stress test velocity field

Smoke Layer

We successfully created a small scenario with our simulation which builds a smoke layer. Figure 13 shows the growing smoke layer. One can see how smoke gradually fills the room from top to bottom.

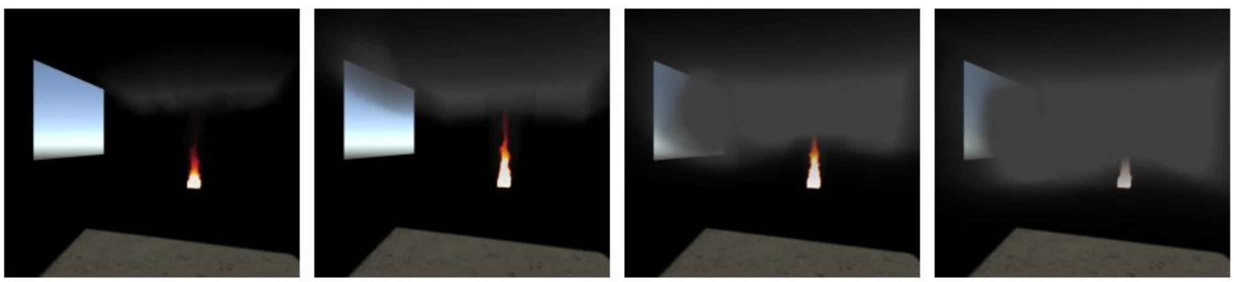

Figure 13: Smoke layer at time frames t=10s, t=90s, t=240s and t=330s

FUTURE PLANS

In the near future we will implement heat transfer from gas to solid, radiation, decomposing objects and the extinguishing process. We already have a radiation model and a working implementation for decomposing objects but it does not fulfill the real-time requirements yet.

As soon as all components are implemented and running in real-time, we will start with reproducing fire phenomena like rollover, dancing angels and flashover. We also are planning to extend the simulation to multiple simulation volumes which are computed on parallel GPUs in one PC.

CONCLUSION

The current simulation using a 256^3 grid runs with 35 frames per second on a NVIDIA TitanXp (Intel i9 2.9Ghz, 64 GB RAM) and with 32 frames per seconds on a NVIDIA GeForce 780Ti (Intel i7 3.6 GHz, 16 GB RAM).

With a cell/voxel size of 2cm in each dimension a volume of $(5.12m)^3$ can be covered. Depending on the use case the voxel size could be smaller or bigger to get more visualization detail or to cover a bigger volume. Pending work like radiation, extinguishing process and decomposing objects will definitely cost performance. But till now we did not spent much time on optimization. Additionally one can reduce the simulation volume to a 128^3 grid, which reduces computing time by factor eight to keep the simulation in real-time. To reproduce the couch burning experiment to its full extend decomposition is necessary, which we are planning for the next version. The stress test already showed the simulation has no performance issues. It should also be possible to reproduce a small room fire even with multiple heat sources and furniture.

REFERENCES

[1] Crane K., Llamas I., Tariq S. (2008), "Real-Time Simulation and Rendering of 3D Fluids", GPU Gems 3, 633-675

[2] Harris M.J. (2004), "Fast Fluid Dynamics Simulation on the GPU", GPU Gems, 637-665

[3] McGrattan K.B., Hostikka S., McDermott R., Floyd J., Weinschenk C., and Overholt K. (2017), "Fire Dynamics Simulator (Version 6) Technical Reference Guide, Volume 1: Mathematical Model", National Institute of Standards and Technology, 6.5.3 edition

[4] Melek Z. (2007), "Interactive Simulation of Fire, Burn and Decomposition", Submitted to the Office of Graduate Studies of Texas A&M University in partial fulfillment of the requirements for the degree of Doctor of Philosophy

[5] Stam, J. (1999), "Stable Fluids", In SIGGRAPH 99 Conference Proceedings, Annual Conference Series, 121-128

[6] Stam, J. (2003), "Real-Time Fluid Dynamics for Games", Proceedings of the Game Developer Conference

[7] Zhao Y., Wei X., Fan Z., Kaufmann A., Qin H. (2003), "Voxels on Fire", IEEE Visualization 2003, 271-278

WEBLINKS

[8] Nakamura Masatatsu, "Voxelizer source code", GitHub (last visit: 27.08.2018) https://github.com/mattatz/unity-voxel

[9] Peel J., Koch G. (2017), "Making it in Unreal: how voxel magic makes the world burn in Firefighting Simulator", PCGamesN (last visit: 27.08.2018) https://www.pcgamesn.com/firefighting-simulator/unreal-engine-4-voxels-fire-propagation

[10] NVIDIA NvFlow (last visit: 27.08.2018) https://developer.nvidia.com/nvidia-flow