

The Vortices Keeper- IVOCK

Integrated Vorticity of Convective Kinematics

A Maya Plug-in Development Tool

By:

Xi Yang

Yi Huang

Based on:

Restoring the Missing Vorticity in Advection-Projection Fluid Solvers

Xinxin Zhang, Robert Bridson, Chen Greif – SIGGRAPH 2015

PROJECT SUMMARY

The goal of our project is to develop a Maya plugin of smoke simulation system. This tool could generate smoke effect using the new scheme IVOCK (Integrated Vorticity of Convective Kinematics) so that the most energy loss in self-advection could be avoided. As most visual effect fluid solvers suffer from the energy loss during the process of projecting self-advection, this tool will capture much of the energy lost in self-advection by identifying it as a violation of the vorticity equation, without using smaller steps or more complex integration which is time-consuming.

It can be used primarily in applications for computer animations and movies. We believe that the tool will provide users to generate satisfying simulation of smoke effects. The user can take control over the elements of the initialization parameters of the grid cells, as well as the time step he/she wants to take. Also, the users are able to change the parameters of the emitter to control the smoke generation, such as temperature and other properties. The users could change the size of each finite difference cell to get the exactly simulation effect that they want. If time permit, the tool will provide more detailed parameter settings to enable further user interaction to modify the smoke simulation process and render effects, such as the boundary limitations, particle and grid properties, as well as the render effects of the smoke.

Our authoring tool will be developed using MEL scripting and C++ API. The core calculation part will be based on the IVOCK method and implemented with C++ API. We may use some parallel structures and resources with C++ program (e.g., INTEL TBB) since this will improve the calculation speed greatly, which will enable this tool for real-time simulation. We will use MEL scripting language to build the GUI for our tool and to deal with some basic tasks, such as the creation and edition of menu, and the edition of Maya Hypershader.

For Alpha version, we will finish the framework and implement a simple grid structure together with a particle system. After this step, we will have a particle system having some fluid features and can be adjusted by Maya UI. The Beta version will add the IVOCK method to generate the vorticity effect. For the Final version, we will add the boundary and end up with an Semi-Lagrangian-IVOCK system.

1. AUTHORING TOOL DESIGN

1.1. Significance of Problem or Production/Development Need

Building animation tools for fluid-like natural phenomena is an increasingly important and challenging problem in computer graphics field. A good solver is of great importance in many different areas. Since the work of Stam in 1999 provides a method by storing variables on a Cartesian grid, there are several solvers developed, such as BFECC method, MacCormack method, FLIP method, etc. However, even the highly accurate advection scheme is used, the self-advection step typically transfer some kinetic energy, losing energy noticeably during the projection for large time steps. Therefore, we aim to build the tool that could avoid most of the energy lost in self-advection, which will be able to upgrade the classic fluid solvers.

Currently, there are several commercially smoke simulation tools for Maya, including some famous tools like FumeFX, Phoenix FD, which are relatively high cost. And the Maya comes with its nParticles which is a particle generation system that uses Maya® Nucleus™ dynamic simulation framework. If possible, we plan to make use of this framework to our authoring tool.

We intend to build a Maya plugin which can solve for the energy loss in self-advection, and can keep a robust and efficient performance.

1.2. Technology

Restoring the Missing Vorticity in Advection-Projection Fluid Solvers,
[Xinxin Zhang, Robert Bridson, Chen Greif, SIGGRAPH 2015].

We mainly build our authoring tool based on the work of Xinxin Zhang et al. This paper provides a new scheme named IVOCK (Integrated Vorticity of Convective Kinematics) which is independent of the advection scheme as well as boundary conditions and other parameters. So, this scheme could be applied on fluids like smoke, fire and water, where the smoke simulation is the most evident (Figure 1). The contribution of this paper includes: IVOCK

scheme to keep energy during self-advection, a set of techniques to store the vortex dynamics on a fixed spatial grid, upgraded classic fluid solvers with IVOCK scheme and a new vortex stretching model. There are other papers of fluid solvers which could be implemented with IVOCK upgrades. In our authoring tool, we will implement the IVOCK method on smoke simulations.



Figure 1- smoke simulation by Xinxin Zhang et al.

1.3. Design Goals

This authoring tool is faced to Maya users. Our tool will provide a realistic simulation of smoke while keep most of the energy during self-advection process. The user will be able to interactively create and modify the emitter and the smoke container. We employ a cheap but efficient method to achieve this goal.

1.3.1 Target Audience.

We build this authoring tool aiming at providing a new smoke solver for animators, modelers, game developer, and other Maya users.

1.3.2 User goals and objectives

With our authoring tool, the users could interactively build a smoke simulation system, and they will be able to create satisfying render results, which we hope to be fast.

1.3.3 Tool features and functionality

The basic elements of our authoring tool are as follows:

Smoke Emitter

The users can generate an emitter of smoke, e.g., the heat source, and are able to edit the properties, e.g., the temperature.

Smoke Container

The user can specify the number of grid cells in each direction of the volume, and control the size of finite difference cell.

Time Step

The user is able to change the time step of the simulation process.

1.3.4 Tool input and output

Input: User setting of parameters listed above.

Output: A 3D simulation of smoke. The output may be a scene in Maya.

1.4. User Interface

The user interface would be a window with buttons and sliders. The GUI would have initial data for generating the smoke.

1.4.1 GUI Components and Layout

The planned GUI is shown as Figure 2. We also add a Menu item shown as Figure 3.

The Menu Item is smoke generator, and it has a sub-item “The Vortices Keeper”. Click this item, the users will enter the setting window of our authoring tool.

As stated in the labels, every parameter can be modified with a slider.

The “create” button will pass the current settings into calculations and generate the wanted smoke effect.

The “cancel” button will close this window.

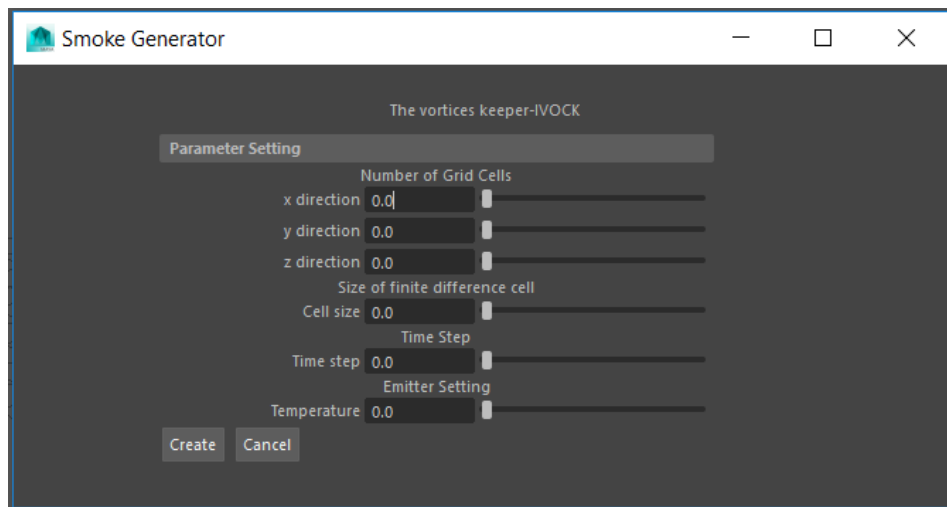


Figure 2- GUI Layout.

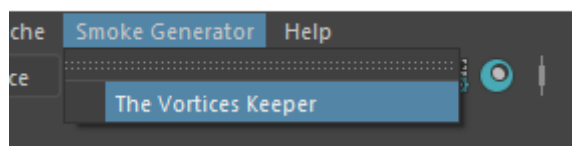


Figure 3- Menu Design.

1.4.2 User Tasks

This authoring tool is interactive and friendly to users. Users just need to set the initial parameters and they are not required to have further knowledge of smoke simulation calculations.

1.4.3 Work Flow

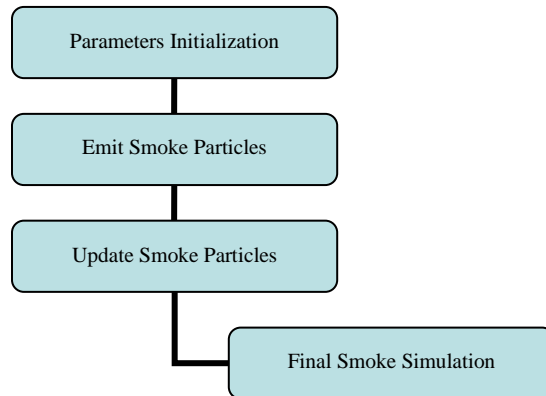


Figure 4- Work Flow.

The work flow is shown as Figure 4. The user will begin by setting the parameters to create a simulation system. And then, those values will be passed into calculations. At every time step, the particles will be updated. A final model will be used to simulate the smoke effect that the user designed.

2. AUTHORING TOOL DEVELOPMENT

2.1. Technical Approach

2.1.1 Algorithm Details

In general, there are two ways to generating fluid simulation, one way is storing fluid variables on a fixed Cartesian grid, which is the Eulerian approach, the advantages of pressure projection on a regular grid is the ease of treating smooth quantities undergoing large deformation.

Another method is vortex methods, although vortex methods is less intuitive working with vorticity rather than velocity in crafting controls for art direction, it can better capture important visual details when running with the same time step and advection scheme.

In this paper, the author combine our observation from vortex methods with Eulerian grid-based velocity-pressure solvers to arrive at a novel vorticity error compensation scheme. The algorithm they come up with is called “IVOCK” (Integrated Vorticity of Convective Kinematics), because they solve vorticity in a violation of the vorticity equation, and measure vorticity on the grid before and after advection, taking into account vortex stretching, and use a cheap multi-grid V-cycle approximation to a vector potential whose curl will correct the vorticity error.

So in rest of this part, we will need to discuss both Eulerian approach and the IVOCK algorithm.

(1) Eulerian approach

In Eulerian simulations, the fluid state is often solved with a time splitting method: given the incompressible Euler equation,

$$\begin{aligned}\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} &= -\frac{1}{\rho} \nabla p + \mathbf{f} \\ \nabla \cdot \mathbf{u} &= 0\end{aligned}$$

fluid states are advanced by self-advection and incompressible projection. First one solves the advection equation.

$$\frac{D\mathbf{u}}{Dt} = 0$$

To obtain an intermediate velocity field $\tilde{\mathbf{u}}$, which is then projected to be divergence-free by subtracting \mathbf{p} , derived by a Poisson solve from itself. We formally denote this as $\mathbf{u}^{n+1} = \text{Proj}(\tilde{\mathbf{u}})$.

But self-advection disregards the divergence-free constraint, allowing some of the kinetic energy of the flow to be transferred into divergent modes which are lost in pressure projection. We focus in particular on rotational motion: self-advection can sometimes cause a noticeable violation of conservation of angular momentum, as illustrated in Fig.1.

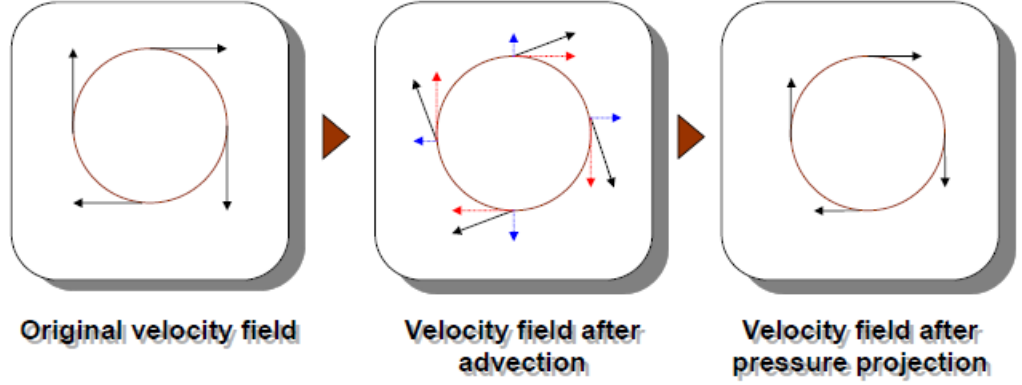


Figure 1: Self-advection maps the original velocity field into a rotational part and a divergent part, indicated by red and blue arrows respectively. Pressure projection removes the blue arrows, leaving the rotational part, and illustrating how angular momentum has already been lost.

(2) The IVOCK scheme

When solving Navier-Stokes, the fluid velocity is usually advanced to an intermediate state ignoring the incompressibility constraint:

$$\begin{aligned}\tilde{u} &= \text{Advect}(u^n), \\ \tilde{\tilde{u}} &= \tilde{u} + \Delta t f,\end{aligned}$$

where u^n is the divergence-free velocity field from the previous time-step and f is a given force field which may include buoyancy, diffusion, vortices confinement, and artistically controlled wind or motion objects.

We observe that in vortex dynamics, the intermediate velocity field \tilde{u} is analogously solved using the velocity-vortices ($u - \omega$) formula:

1. given ω^n , solve

$$\frac{D\omega}{Dt} = \omega \cdot \nabla u$$

to get $\tilde{\omega}$;

2. deduce the intermediate velocity field \tilde{u} from $\tilde{\omega}$.

The above equation suggests the post-advection vorticity field $\omega^* = \nabla \times \tilde{u}$

should ideally equal the stretched and advected vorticity field ω , but due to the simple nature of self-advection of velocity ignoring pressure, there will be an error related to the time step size.

We therefore define a vorticity correction

$\delta\omega = \tilde{\omega} - \omega^*$ from which we deduce the IVOCK velocity correction δu and add this amount to the intermediate velocity \tilde{u} . Algorithm 1 provides an outline of the IVOCK computation before we discuss the details.

Algorithm 1 IVOCKAdvection($\Delta t, u^n, \tilde{u}$)

- 1: $\omega^n \leftarrow \nabla \times u^n$
 - 2: $\tilde{\omega} \leftarrow \text{stretch}(\omega^n)$
 - 3: $\tilde{\omega} \leftarrow \text{advect}(\Delta t, \tilde{\omega})$
 - 4: $\tilde{u} \leftarrow \text{advect}(\Delta t, u^n)$
 - 5: $\omega^* \leftarrow \nabla \times \tilde{u}$
 - 6: $\delta\omega \leftarrow \tilde{\omega} - \omega^*$
 - 7: $\delta u \leftarrow \text{VelocityFromVorticity}(\delta\omega)$
 - 8: $\tilde{u} \leftarrow \tilde{u} + \delta u$
-

For implementing algorithm 1, as for the data storage, Extending the classic MAC grid widely adopted by computer graphics researchers for fluid simulation, we store vorticity and stream function components on cell edges in a staggered fashion, so that the curl operator can be implemented more naturally as illustrated. For example, in figure 2, if the grid cell size is h , the z-component of vorticity on the z-aligned edge centered is approximated as

$$\omega_z(i, j, k + \frac{1}{2}) = \frac{1}{h} \left[\left(v(i + 0.5, j, k) - v(i - 0.5, j, k) \right) - \left(u(i, j + 0.5, k) - u(i, j - 0.5, k) \right) \right]$$

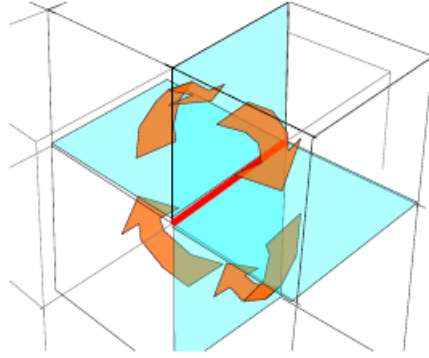


Figure 2: *Vorticity and stream function components are stored in a staggered fashion on cell edges in 3D (red line), while velocity components are stored on face centers. This permits a natural curl finite difference stencil, as indicated by the orange arrows.*

For the vortex stretching and advection in grid, when discretizing equation on an Eulerian grid, a geometrically based choice would be to compute the update with the Jacobian matrix of u :

$$\omega^{n+\frac{1}{2}} = \omega^n + \Delta t \mathcal{J}(u) \omega^n.$$

2.1.2 Maya Interface and Integration

Paint Tool:

The painting tools will be scripted mainly using MEL. We will need to make use of a plug-in programmed in C++ through the Maya API in order to store information about our environment and the fuel cells in a data structure.

Particle up to date module:

This module uses the environment data structure that was created via the paint tool, and will transfer its renewed data structure to the fluid solver. This part will use the algorithm we discussed above to create the kinematic information. We need to implement the grid structure as well as the IVOCK here. This part will be written in C++ API.

Particle render solver:

The Fluid solver will be written in C++ API, which is time-step and can updates a whole fluid step. It should also have an emitter function that trace particles and indicates smoke source. We may use Maya's particle functions within this part, but this is still to be determined. In order to render the final results in Maya, the data-type need to be transferred to a structure accepted by Maya.

2.1.3 Software Design and Development

The main blocks are as follows.

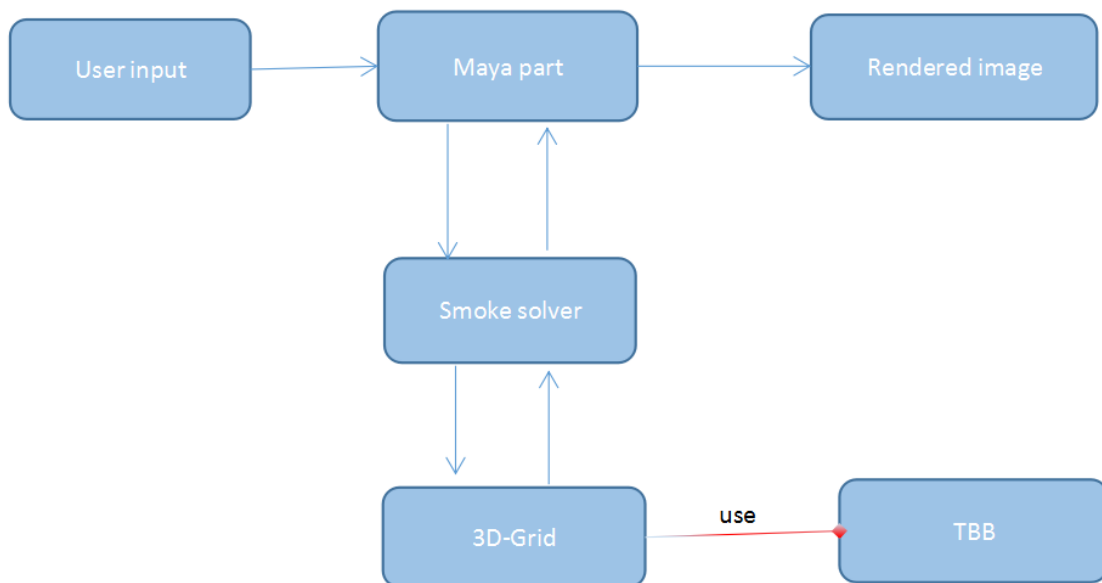


Figure 3 main blocks

The Maya part needs to collect and transfer the data from user input to the smoke solver in a c++ format and transfer the data

from the smoke solver back to Maya and find a way to render it.

The Smoke solver part contains the methods and algorithm dealing with time steps and other particle information. The IVOCK is realized here.

The 3D-Grid part includes the grid data structure and the grid controller, and will use the Intel TBB, which can use to make multi-threads to make the program run faster.

We plan to use the following data structures and classes to share information within the system.

Fluid_particle class: A class to simulate the particle. Stores information like its velocity, position density temperature and the index. It uses the time step to update the latest information of the property of the particle class, like the velocity, position, temperature and so on.

Array3: A class stores the i,j,k value, which can be used to store and transfer 3 dimensional data like position and velocity in this whole system to help transferring values .

Smoke solver class: A class that can be used to mimic the smoke behavior, which includes the tolerance, iteration, boundary solver and so on. The IVOCK methods introduced in the paper will also be implemented in this part and combined with other code.

PCGSolver class: Implements PCG with Modified Incomplete Cholesky (0) preconditioner. PCG Solver<T> is the main class for setting up and solving a linear system. Note that this only handles symmetric positive (semi-)definite matrices, with guarantees made only for M-matrices .

MultiGrid class: This class is going to create multiGrid. given A_L , R_L , P_L , b , compute x using Multigrid Cycles.

fluidBuffer class: This class is going to create a 3D buffer which can store 3 dimensional data which is in need. It also sets the linear and cube methods to compute the grid.

SparseMatrix class: This class is used to dynamically compress the sparse row matrix.

2.2. Target Platforms

2.2.1 Hardware

- (64-bit) Intel® Pentium® I5 or higher
- 2 GB RAM or higher
- Graphic cards: NVIDIA GTX960M

2.2.2 Software

- (64-bit) Windows 10
- Maya 2016
- Microsoft Visual C++ 2010 or higher

2.3. Software Versions

2.3.1 Alpha Version Features (first prototype)

The Alpha version will have 3 main implementation, one is the framework. In this part, we need to create a user paint tool which contains an UI. It will be capable of generating a data structure containing information about the environment, e.g. the GridX,GridY,GridZ, the time step the size of each finite difference cell and the temperature of the heat source.

Then we need to initialize our simple grid structure. We should also implement a particle system in this version which can simulate some fluid like behavior and can be returned to Maya in a data structure Maya can handle with. Perhaps we will use Maya's particle functions.

2.3.2 Beta Version Features

In this version, we will use the algorithm illustrated in the paper, add the IVOCK method to our system to generate the vortices. In this step, we basically focus on create a Semi-Lagrangian-IVOCK for our project.

2.3.3 Description of any demos or tutorials

In order to make our project friendlier to other users, we will provide a tutorial and a demo. The demo will show the outcome of our smoke generated using the IVOCK method, and how the smoke will change if we adjust the parameters. The tutorial will not include the algorithm since our plan is not to teach the mathematics problems. Instead, we will cover things like why this plugin is useful, what can we achieve using this plugin, how to install our plugin. We will also illustrate each parameters meaning, how can they influence the final outcome and will cover some figures to explain this.

3. WORK PLAN

3.1. Tasks

We had divided our project into 7 distinct tasks

Task 1: Framework 10 days Xi Yang

Create the framework of our vortices project. In this part, we need to create a user paint tool which contains an UI. It will be capable of generating a data structure containing information about the environment, e.g. the GridX, GridY, GridZ, the time step the size of each finite difference cell and the temperature of the heat source.

Task 2: Grid structure 10 days Yi Huang

Create a grid structure. In order to handle the values we transferred into our system, we need to create a grid structure to store the fluid variables. The variables stored in this structure can be updated automatically after each time step.

Task 3: Particle system 8 days Xi Yang

We need to create a particle system to simulate the particles using the information we inputted by ourselves. We need to find a way to transfer this data to a way that Maya can render these particles. For this task, we only need to use small spheres to represent the smoke particles.

Task 4: TBB Framework 8 days Yi Huang

Threading Building Blocks (TBB) is a C++ template library developed by Intel for parallel programming on multi-core processors. Using TBB, a computation is broken down into tasks that can run in parallel. The library manages and schedules threads to execute these tasks. We plan to use TBB to make our rendering more effective, for this task, we need to set some configuration parameters and build a grid framework based on TBB.

Task 5: IVOCK method 14 days Yi Huang

For this task, we need to use the algorithm illustrated in the paper, add the IVOCK method to our system to generate the vortices. In this step, we basically focus on create a Semi-Lagrangian-IVOCK for our project.

Task 6: Smoke solver 14 days Xi Yang

For this task, we need to use the algorithm illustrated in the paper to create our smoke solver.

Task 7: Boundary buffer check 10 days Yi Huang

For this part, we need to fill the boundary will the voxel data created before. To do this, we can either us C++ to write our own methods or try to use Maya interface to specify the solver domain and other parameters.

Task 8: Smoke effect rendering 10 days Xi Yang

For this part, we need to find some way to render the particle to make it looks like smoke. We may choose either Maya or C++ code to achieve this goal.

Task 9: Testing/Debugging 10 days Yi Huang/ Xi Yang

This task will be general testing and debugging of the integrated plug-in to see if something else needs to be added to the system and to improve the behavior of the project.

Task 10: Writing tutorial 3 days Yi Huang/Xi Yang

In order to make our project more friendly to other users, we will provide a tutorial. We will cover things like why this plugin is useful, what can we achieve using this plugin, how to install our plugin. We will also illustrate each parameters

meaning, how can they influence the final outcome and will cover some figures to explain this.

3.1.1 Alpha Version

This version we plan to accomplish task 1,2,3,4

3.1.2 Beta Version

This version we plan to accomplish task 5,6

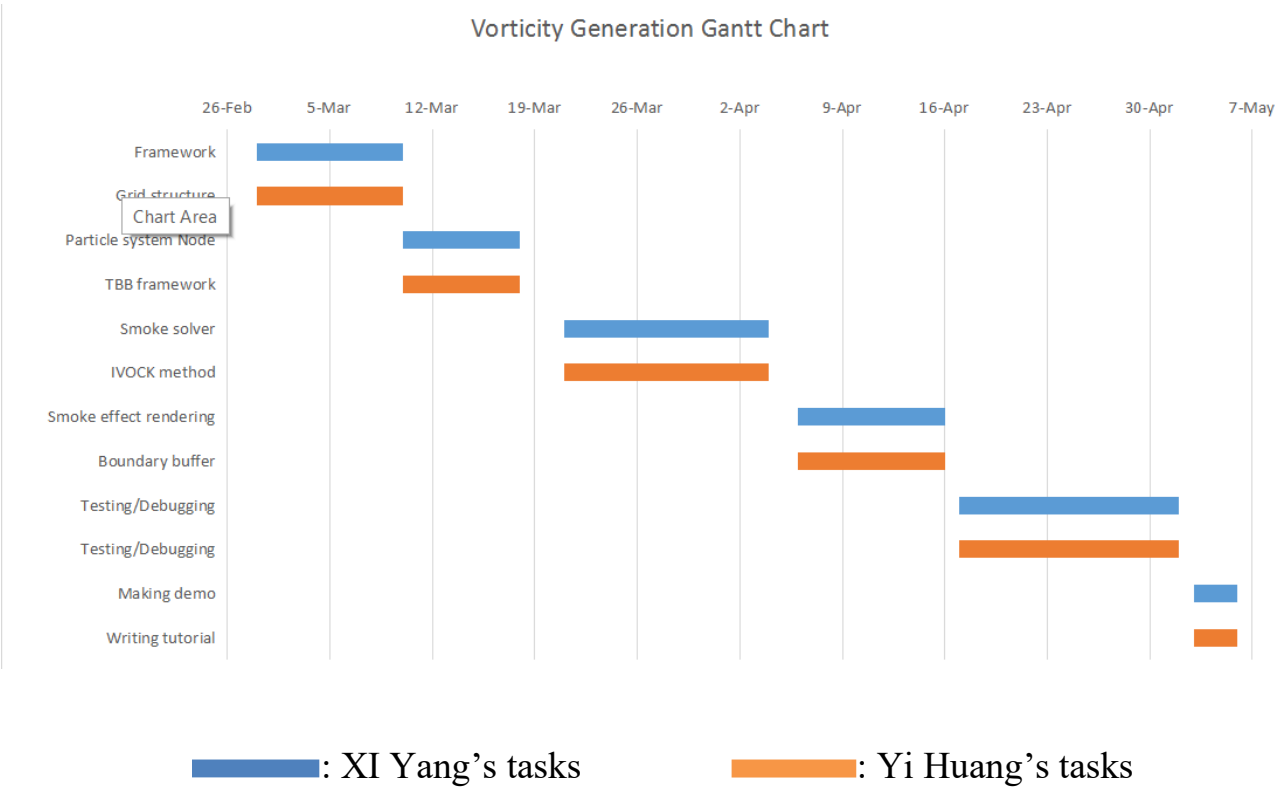
3.1.3 Final Version

This version we plan to accomplish task 7,8,9,10

3.2. Schedule

The Gantt chart is showing as bellow:

The bars in blue represent Xi’s tasks and the bars in orange are Yi’s tasks.



4. Results

4.1 Tutorial

Our authoring tool is a Maya plugin, so in order to use our tutorial, you should first install our plugin like any other Maya plugin.

Figure 4 shows the workflow of the Vortices Keeper.

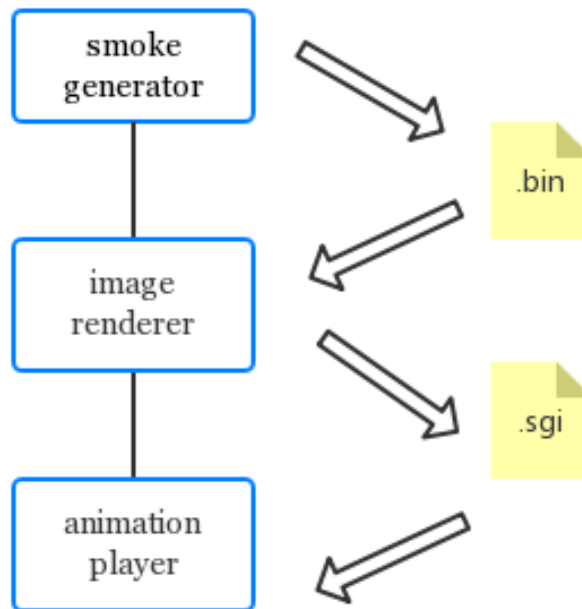


Figure 4

The Vortices keeper has three parts, a smoke generator, an image renderer and an animation player. The smoke generator determines the position of each particle in the smoke system; the image renderer uses the data information created by the smoke generator and will generate 3D images based on this information; the animation player can play the 3D images in animation directly in Maya.

Basically, after installed the plugin, there are 3 steps to use it.

Step 1:

Click “the Vortices Keeper” → “smoke generator”, set the parameters to create the original particle systems.

Step 2:

Click “the Vortices Keeper”→ “render image”, set the parameters to generate the 3D images.

Step 3:

Click “the Vortices Keeper”→ “display animation”. To see the final outcome, right click “file” → “Open Animation” to see the render effect.

4.2 Content Creation

The UI for Smoke Generator shows in figure 5, and the list below shows the meaning for each parameter and the effect when changing them.



Figure 5

Advection type:

Shows the solver you choose to generate the smoke system. There are 8 different types to choose, 4 are original solvers without the IVOCK method while others are with the IVOCK method.

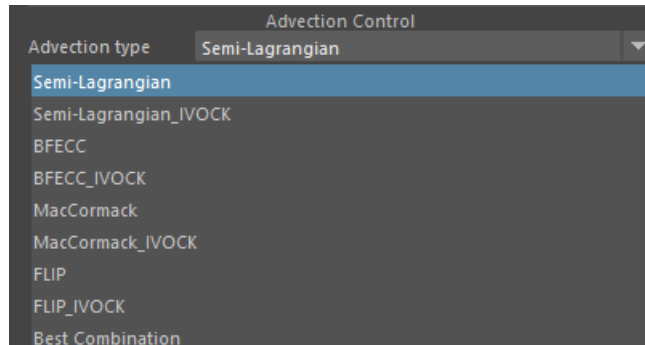


Figure 6

Vortices confine:

Defines the intensity of the vortices and ranges from 0 to 1.

Emitter position:

Defines the original position of the particle system.

Wind force:

Defines the wind force of the particle system if exists.

Number of grid cells:

Defines the size of the particle system in certain direction. The bigger the grid size is, the slower the generation. 32 X 64 X 32 is the default set and 128 X 256 X 128 is the max size Maya can bear.

Cell size:

Defines the size of each particle.

Temperature:

Defines the temperature of current system, the higher the temperature, the faster the smoke will arise. The default value of this parameter is 500.

Smoke density:

Defines the density of the smoke system. 10.0 is the original value.

Alpha:

Defines the buoyancy of the smoke, which ranges from 0 to 1 and the default value is 0.15.

Beta:

Defines the rate for the smoke which falls to dust in each step. The default value is 0.25 and it ranges from 0 to 1.

Max frame:

Defines the number of frames generates by the particle system, the default value is 60 and it ranges from 0 to 240.

Output file path:

Defines the path for the plugin to store temporary data.

After the parameters are set, click create, wait for a while and Maya will generate a particle system like showing in figure 6.

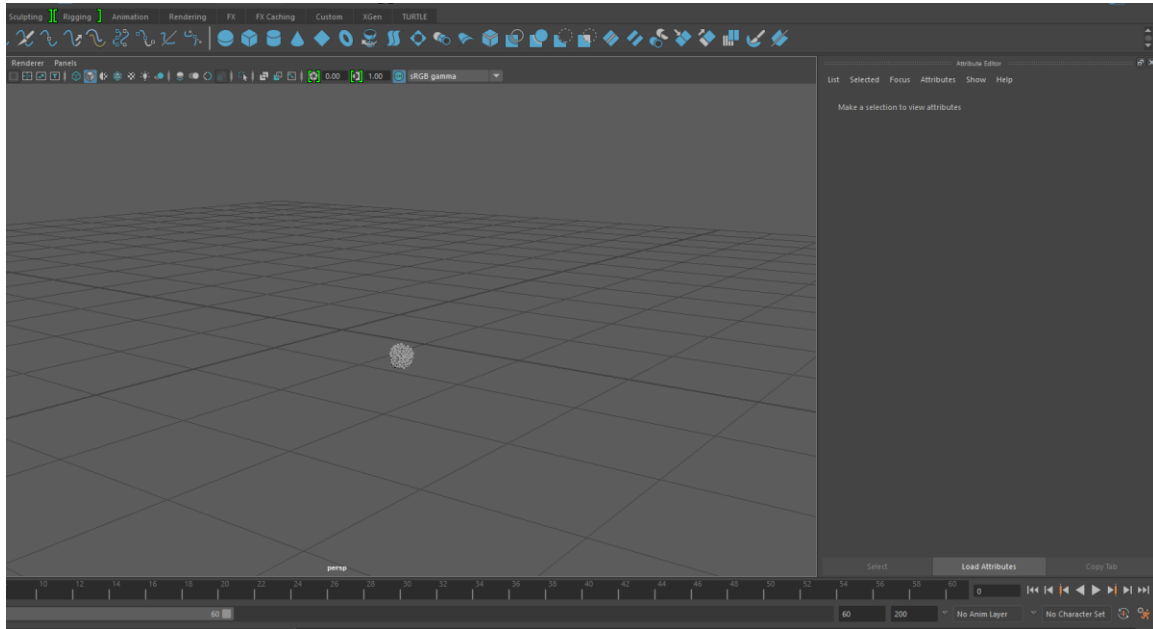



Figure 7

By clicking the buttons  on the right, you can show the general shape of the smoke directly in Maya and can play back and forward. You will get a result like showing in figure 7 during this process.

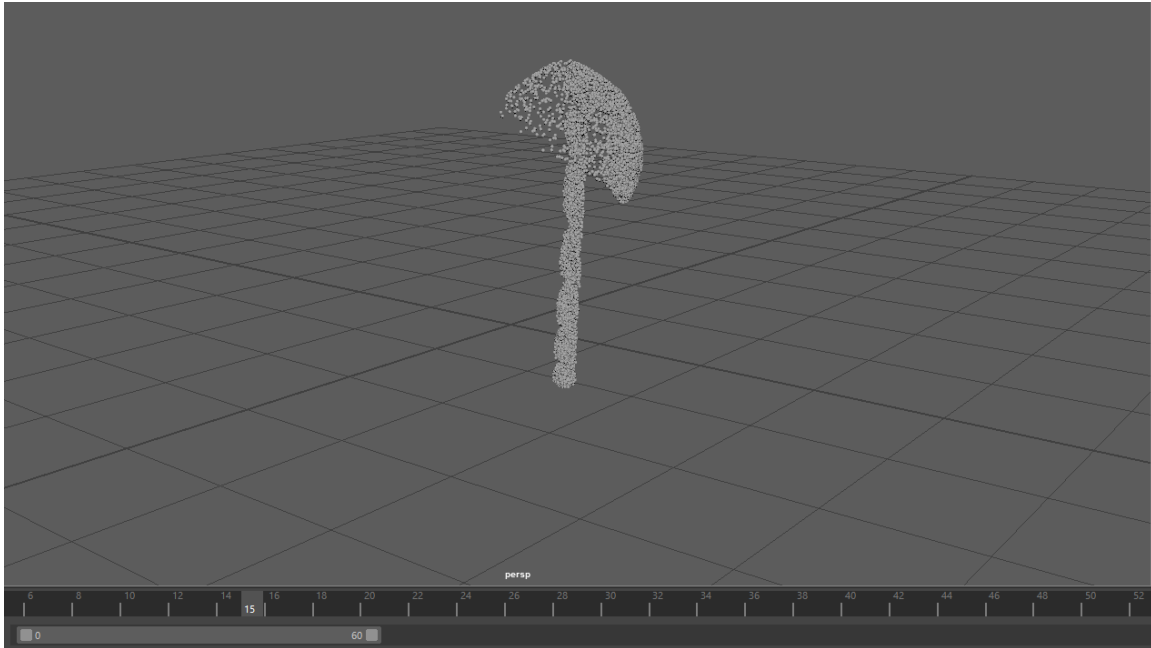


Figure 8

The Render Image UI is showing in figure 8. This figure shows the default setting of each parameter, which can be adjusted to get more different rendering effects.

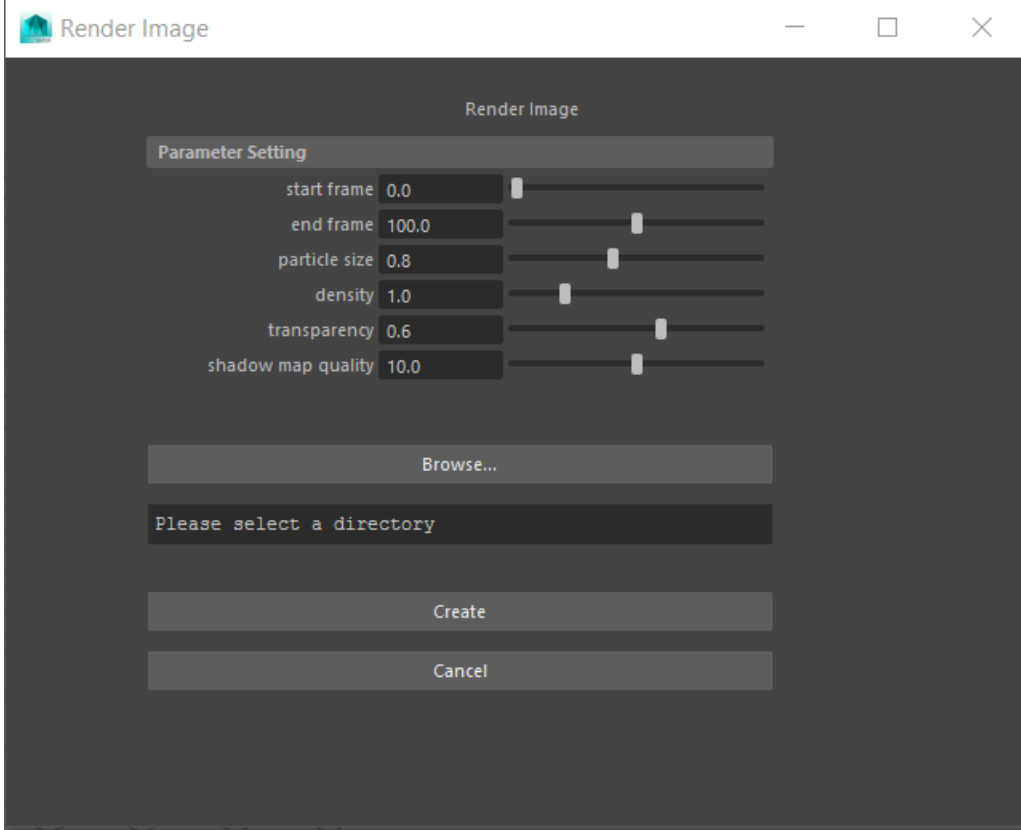


Figure 9

After rendering, click “Display Animation” to view the smoke effects. By click “file” → “Open Animation” and click the play button, you will see the smoke effects. Figure 9 is an example.

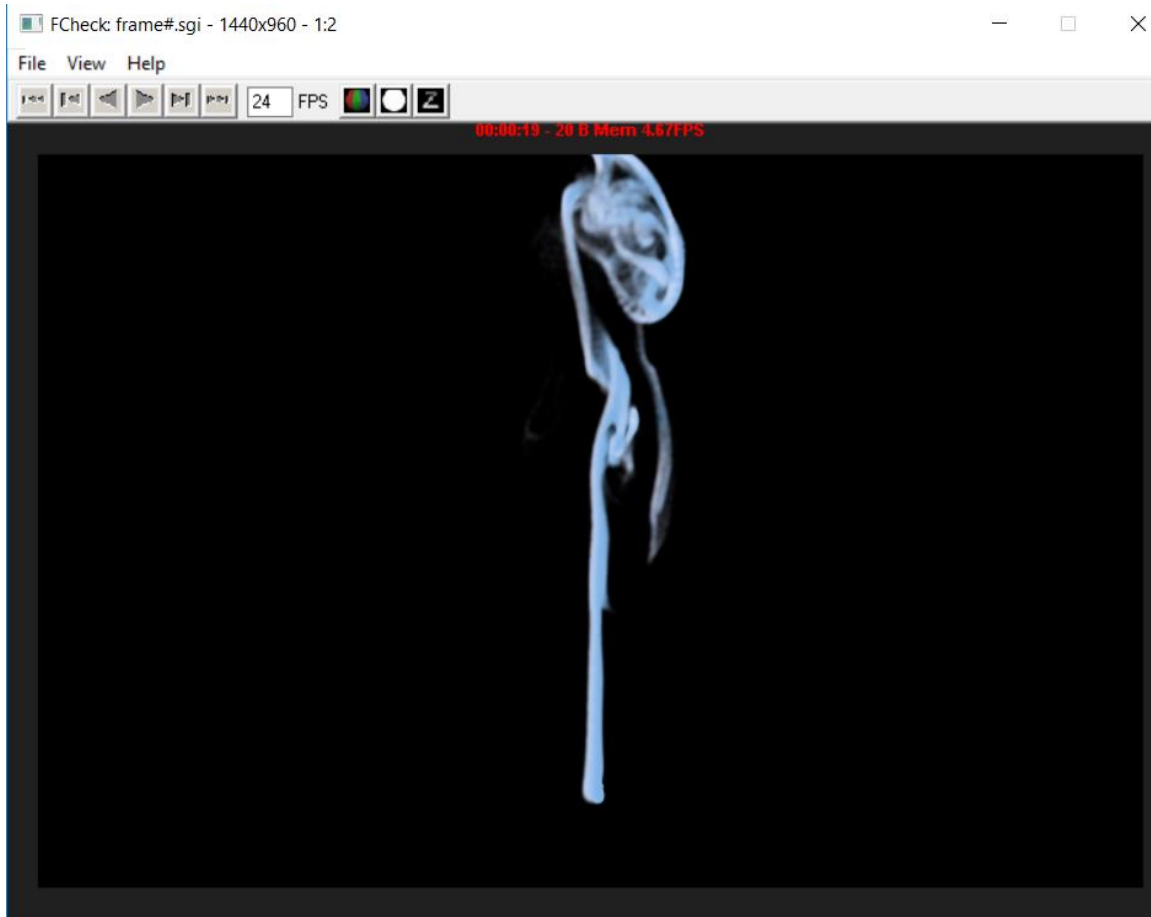


Figure 10

4.3 Discussion

4.3.1 Accomplishments

In general, we had achieved most features we described before.

Implemented features:

- A smoke particle system to generate particles in Maya.
- A renderer to show the 3D rendering effect.
- An outcome animation can be directly viewing in Maya.

We also added features which was not described in the design doc, these features are:

- Different solvers for user to choose when generating smoke.
- Additional forces (wind force) to rich the animation.

Features not implemented:

As we described earlier, we wanted to generate the smoke at real time, finally we found this was really difficult due to the heavy computation for generating smoke.

4.3.2 Design Changes

First, as described before, due to the heavy computation for generating smoke, instead of trying to generate each frame of the smoke at run time, we compute all of these frames when initialize the system. This makes the effects become more clearly and the play back and forward more easily. And the time cost for a small initialization is acceptable, only for 1-2 minutes.

Second, instead of writing the solver by ourselves as we planned, we used the existing solvers provided by the author and focused mainly on realizing the vortices method and the Maya part. This change gives us enough time to focus on things that are more important.

4.3.3 Total Man-Hours Worked.

During the past 10 weeks, the total man-hours worked and each members' working hours are listed below.

Yi Huang: 107 hours

Xi Yang: 117 hours

Total: 224 hours

4.3.4 Future Work

Right now, the only forces the author can have control are the vortices and the wind force, so in the future, we will add more different types of forces for the authors to explore. In addition, right now the renderer can only provide one color of smoke, in the future, we want to add more rendering effects to this plugin. Also, right now the plugin can only generate one

smoke at a time, in future work, we will make it be able to generate several smokes at the same time and to see their interactions with each other.

4.4 Third Party software

In order to accomplish the work, we used several third-party software and received a lot of help from the author. These materials are listed as below:

Third party software:

Threading Building Blocks (Intel® TBB)

Intel® Threading Building Blocks (Intel® TBB) lets you easily write parallel C++ programs that take full advantage of multicore performance, that are portable and composable, and that have future-proof scalability.

Fluid solvers

All the original fluid solvers used in this project are provided by the author, they are C++ code.

Smoke renderer

The smoke renderer we used to render the smoke was given by the author of the SigGragh paper we chose.

4.5 Lessons Learned

- Be suspicious when using the examples in Maya documentation. Some instructions to use were not included, some examples are wrong. For example, the documentation for the nParticle.
- There can also be some problem with Maya build-in plugins when using the example code, such as particleAttrNode and simpleEmitter.
- The script editor was really helpful when you don't know how to use some Mel commands. You can do the operation manually and the Mel command will show in the script editor.
- If you install the plugin built in debug mode, the speed will be obviously slowed down.

5. RELATED RESEARCH

Some most related and valuable papers are listed below:

- [S99] STAM, J. 1999. Stable fluids. In Proc. ACM SIGGRAPH, 121 – 128.
- [KLLR05] KIM, B., LIU, Y., LLAMAS, I., AND ROSSIGNAC, J. 2005. Flow-Fixer: Using BFECC for fluid simulation. In Proc. First Eurographics Conf. on Natural Phenomena, NPH’ 05, 51 – 56.
- [SFKLR08] SELLE, A., FEDKIW, R., KIM, B., LIU, Y., AND ROSSIGNAC, J. 2008. An Unconditionally Stable MacCormack Method. J. Scientific Computing 35, 2 – 3, 350 – 371.
- [ZB05] ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. ACM Trans. Graph. (Proc. SIGGRAPH) 24, 3, 965 – 972.
- [ZBG15] ZHANG, X., BRIDSON, R., AND GREIF, C. 2015. Restoring the missing vorticity in advection-projection fluid solvers. ACM Trans. Graph. 34, 4 (July), 52:1 – 52:8.
- [HW65] HARLOW, F., AND WELCH, J. 1965. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. The Physics of Fluids, December 1965, 8:2182 – 2189.
- [FM96] FOSTER N., METAXAS D.: Realistic animation of liquids. Graphical Models and Image Processing 58, 5 (1996), 471 – 483.
- [PK05] PARK, S. I., AND KIM, M. J. 2005. Vortex fluid for gaseous phenomena. In Proc. Symp. Comp. Anim., 261 – 270.
- [WP10] WEISSMANN, S., AND PINKALL, U. 2010. Filament-based smoke with vortex shedding and variational reconnection. ACM Trans. Graph. (Proc. SIGGRAPH) 29, 4, 115.
- [PG12] PFAFF, T., THUEREY, N., AND GROSS, M. 2012. Lagrangian vortex sheets for animating fluids. ACM Trans. Graph. (Proc. SIGGRAPH) 31, 4, 112:1 – 8.
- [SU94] STEINHOFF, J., AND UNDERHILL, D. 1994. Modification of the Euler equations for “vorticity confinement” : Application to the computation of interacting vortex rings. Physics of Fluids 6, 2738 – 2744.
- [GLG95] GAMITO M. N., LOPES P. F., GOMES M. R.: Two-dimensional simulation of gaseous phenomena using vortex particles. In the 6th Eurographics Workshop on Computer Animation and Simulation (1995), vol. 28, Springer-Verlag, pp. 3 – 15.
- [SDT08] STOCK, M., DAHM, W., AND TRYGGVASON, G. 2008. Impact of a vortex ring on a density interface using a regularized inviscid vortex sheet method. J. Comp. Phys. 227, 9021 – 9043.
- [MSJ08] MÜLLER, M., STAM, J., JAMES, D., AND THUEREY, N. 2008. Real time physics: class notes. In ACM/SIGGRAPH classes, 1 – 90.
- [SF95] STAM J., FIUME E.: Depicting fire and other gaseous phenomena using diffusion processes. In Proceedings of SIGGRAPH 1995 (1995), pp. 129 – 136.

[CK98] COTTET G.-H., KOUMOUTSAKOS P. D.: Vortex Methods: Theory and Practice. Cambridge University Press, 1998. Book: FLUID SIMULATION SIGGRAPH 2007 Course Notes, Robert Bridson.