Remake

build system
for animation projects

Konstantin Dmitriev
ksee.zelgadis@gmail.com

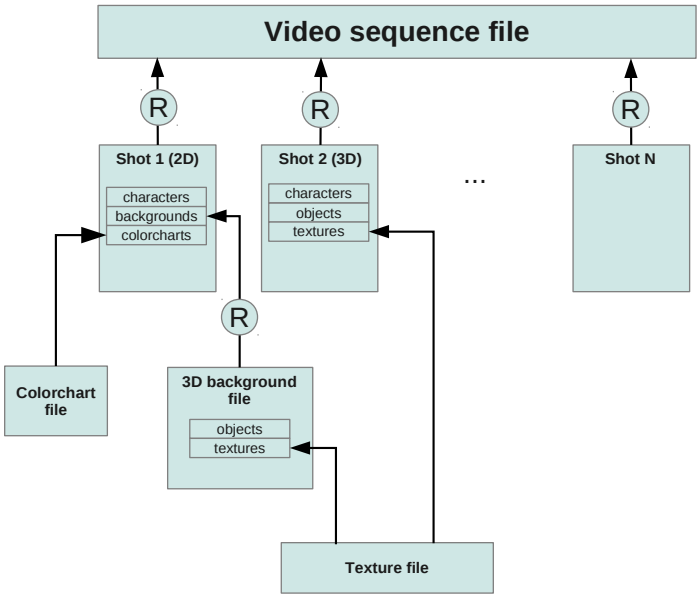
Hi! My name is Konstantin Dmitriev and I'm here to talk about Remake.

Agenda

- The Problem
- How Remake works
- Practical example
- Usage cases
- Development perspectives

Remake is an utility that helps you to automate rendering routines. Let's begin with a question why do you need Remake when you working on animation project.

The Problem



Typical animation project consist of many files, each related with other. Let's look at the illustration. Project is divided into shots (or scenes or whatever you call it). And all they are composed in the video sequence file.

So, here's the main file.

And to be inserted into sequence file each shot should be **rendered**.

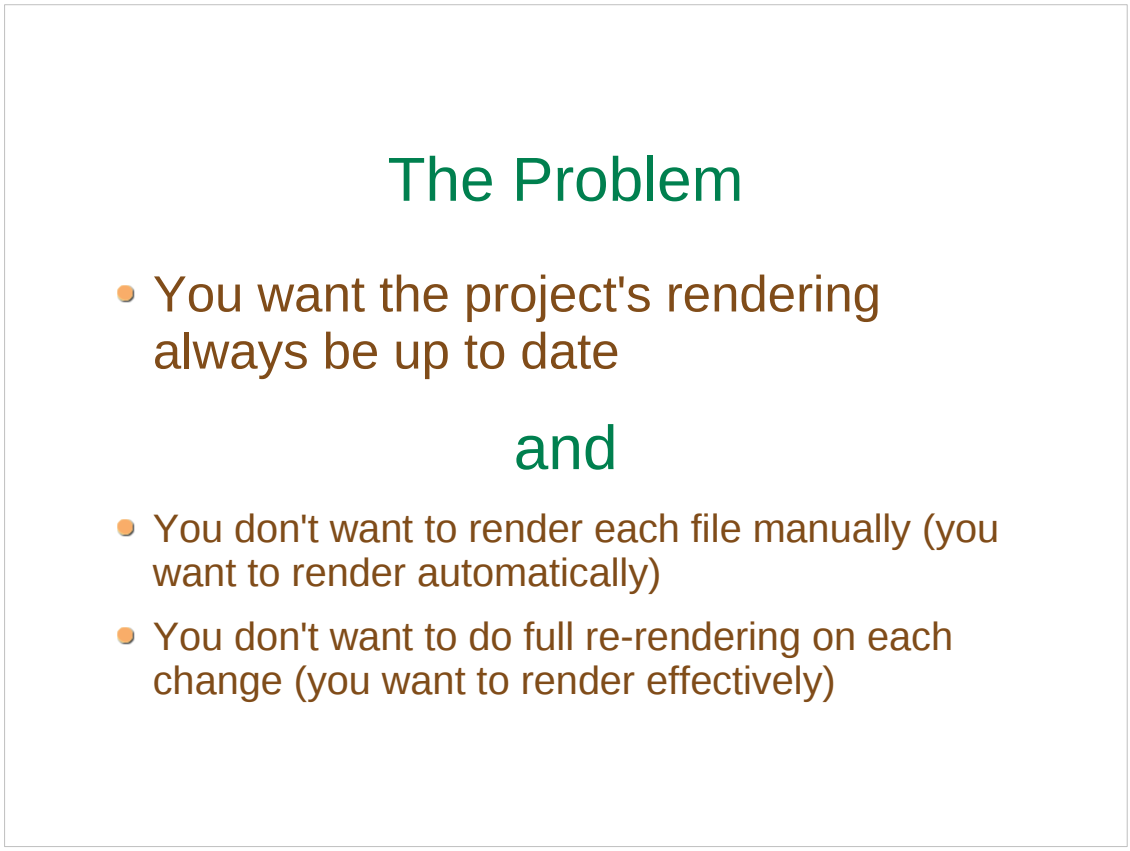
Each shot have several assets – like colorcharts, textures, models, etc.

Those assets could be stored in the shot file itself, or reside in separate files.

Or even more, maybe we can have 2D shot including 3D background. which could be rendered and have his own assets – textures, objects... Maybe some assets are shared with other shots...

As you can see, the whole picture can be quite complex. Consider that typical 4 min long animated short consist of approximately 50 shots.

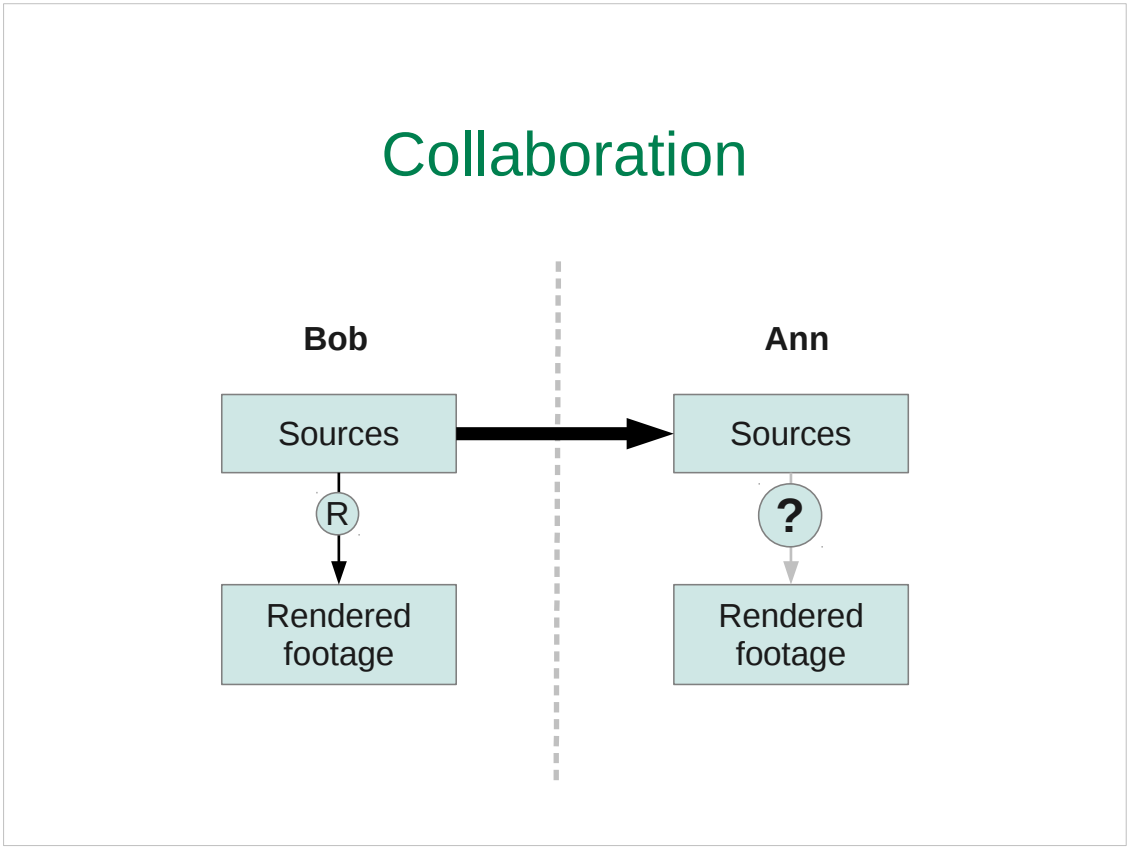
It's obvious that even a tiny change can influence the whole project and you always need to have the whole picture up to date.



So here's the problem – you have a lot of files with complex dependencies and changing one file may require re-rendering of other files (changes are indirect). And you always want the rendering of your project to be up to date.

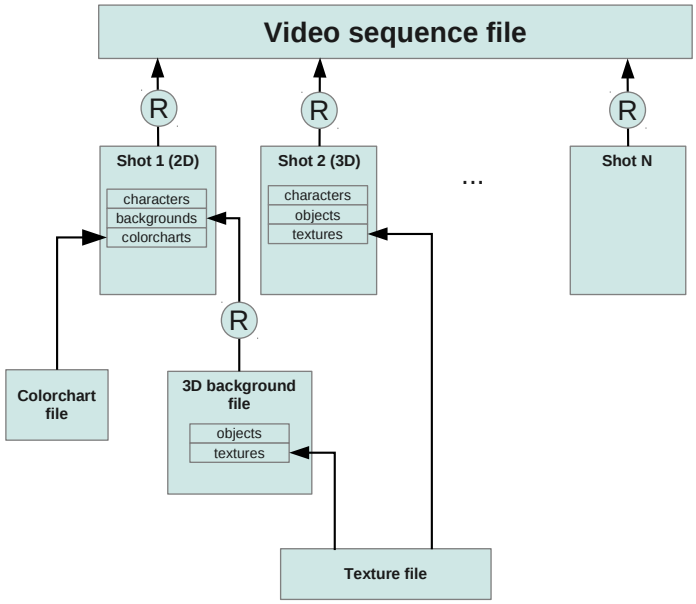
At the same time you don't want to render each file by hand, you don't want to dig into all dependencies each time you want to update rendering – so you want this process to be as automatic as possible.

And you want the effective rendering – we don't want to render all project files every time, we want to re-render only files affected by the change.



If you collaborating on the project with someone online, then it most likely that you will want to minimize the amount of transferred data. And the most effective solution is to transfer source files only. That means that the person you collaborating with will have to render sources himself and he will face the same problem.

The Problem



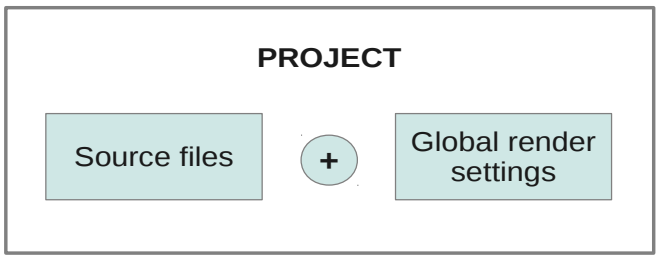
More than that, in case of multiple people collaborating on the project the problem becomes even worse, because it comes to tracking changes. Modern synchronization tools and VCS's provide a good way to track changed files. But knowing the changed file doesn't rescue. Even if we know which files were changed, to re-render/update the project we need to know how changed files related with each other (see illustration). Where and how each file used. What is the place of each file in the whole structure.

It is good if it is YOU who created the shot, who defined the shot structure. But if not – it could be quite tricky to dig into each shot and figure out relationships each time you need to update your project rendering

And remember, we are talking about the project in development. As practice shows, the structure of single shot , all those relations can change dramatically a few times during the development process.

And so, you end up in the situation – everytime someone updates the project, the others are scratching their heads – what I should re-render to get the project up to date?

The Project concept



And finally, when we talking about re-rendering problem it becomes obvious that we need some rendered parameters controlled globally. And this lead to concept of a “project”, a single unit containing all the files and sharing common parameters for rendering.

For example, at the first stages of animation development you can work on the small resolution to optimize rendering speed and have faster updates. And when it comes to final render you should be able to switch to higher resolution with minimal efforts. Or at the last moment client asks you to deliver resulting work in different resolution -twice higher than original. Situation is the same – you need to quickly switch resolution and re-render.

Of course you may write some scripts that store your resolution in variables and do rendering of your files. But remember, our project's structure is continuously changes. So you will end up in continuous changing your scripts. Or you will start writing more smart scripts that consider file dependencies. And then you end up with the same thing that I did – you will write Remake.

But hey, since it's already here let's take a look at it.

How Remake works

1. File analysis
2. Generation of Makefile
3. Make invocation

Remake tracks dependencies in your project and depending on the changed files executes necessary operations to keep your rendering up to date. Let me say it again: it doesn't render the whole project every time. Only the files affected by the change are re-rendered.

How does it technically works. To track changes in the files Remake uses GNU Make utility.

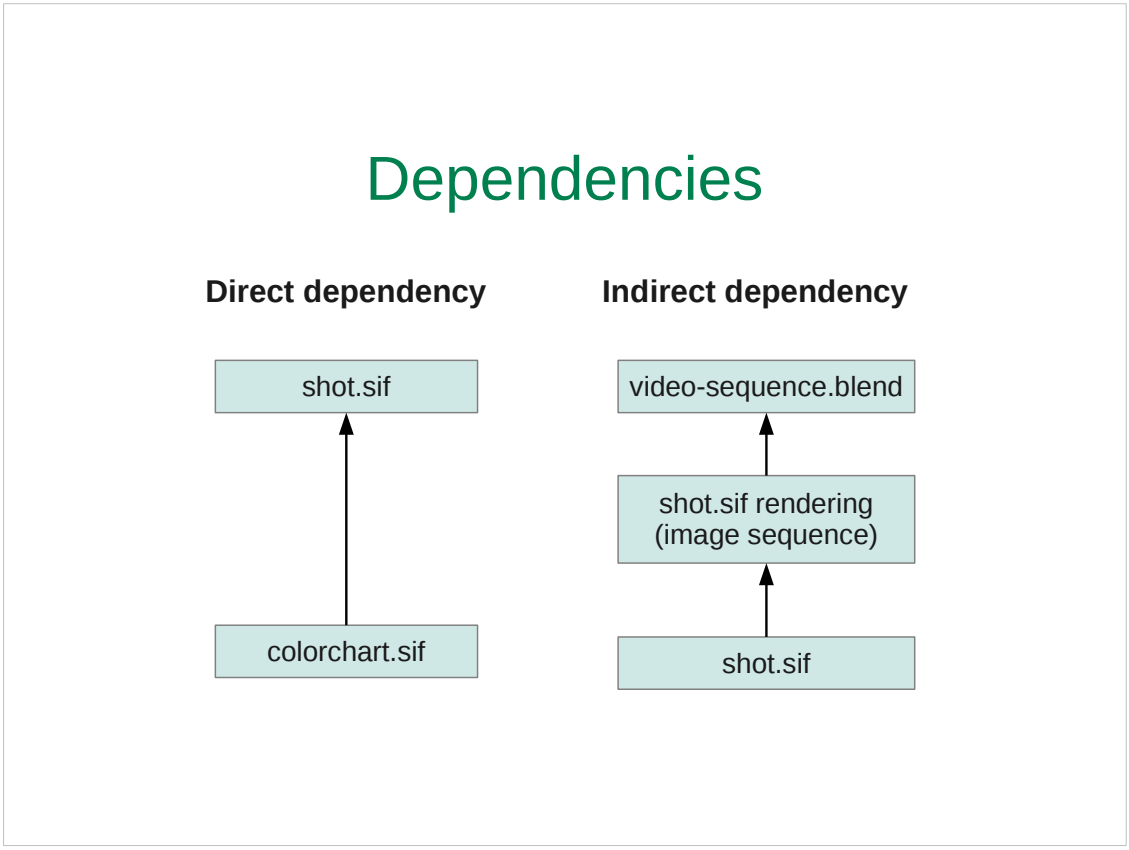
First it analyses file (given to render) and all its dependencies. Then it generates Makefile with all necessary rules. And finally it invokes Make to track changed files and carry all necessary operations.

Supported software

- Blender (3D animation, video editing)
- Synfig Studio (2D animation)
- Pencil (2D animation)

Remake have modular architecture and the list of supported applications can be easily extended by writing additional modules.

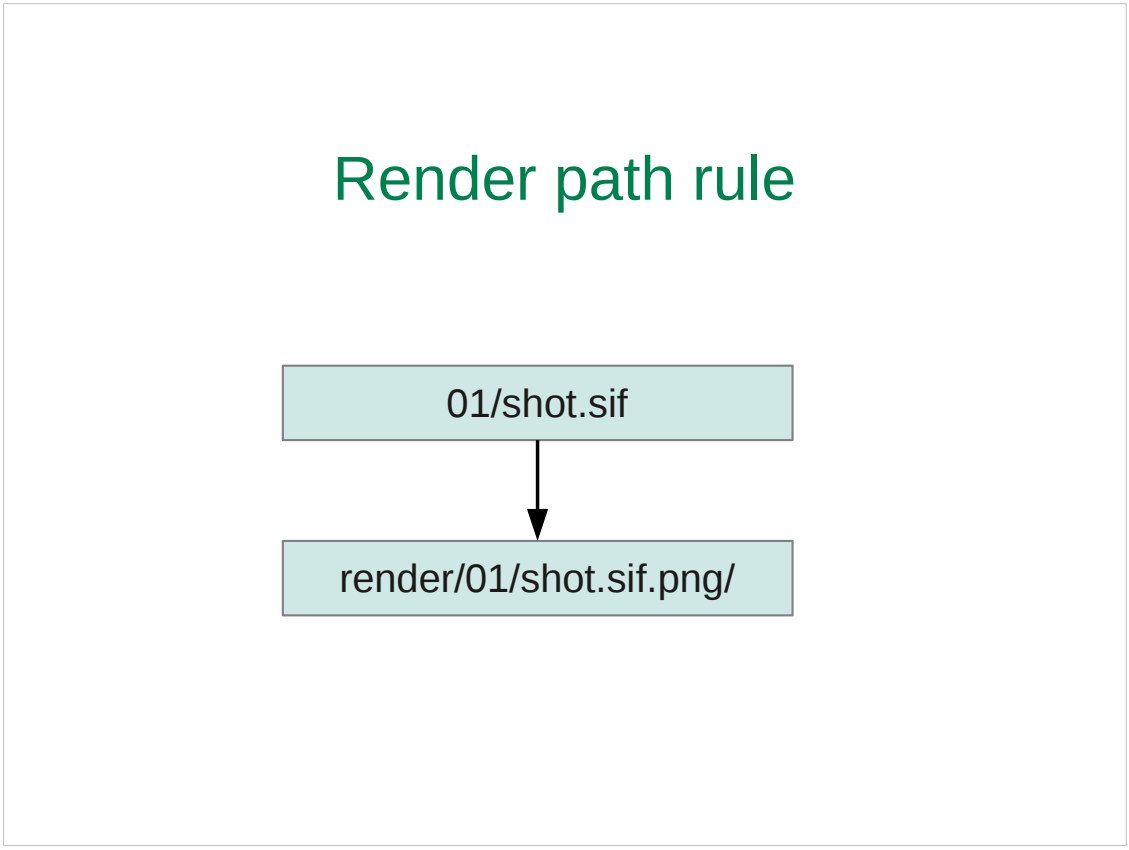
Currently supported software is Blender, Synfig Studio and Pencil (for Pencil our patches are required).



Remake recognizes two types of dependencies.

First type is direct dependency – it means that one file directly included to another. In this example shot.sif directly depend on colorchart.sif. When we try to render shot.sif remake will check if both files were changed since the last rendering and do re-rendering if some of them was.

There's also indirect dependencies. For example shot.sif cannot be directly inserted into video-sequence.blend, because sif files are not supported by Blender. So we need to render sif file into image sequence first and then insert rendered sequence into .blend file. Thus if we will ask Remake to re-render video-sequence.blend again, then Remake will detect that it depend on image sequence, which is the rendering of shot.sif. If shot.sif was changed, the it will be re-rendered and that will trigger re-rendering of video-sequence.blend. If shot.sif wasn't changed since the last rendering, but video-sequence was then only video-sequence will be rendered. If both files are unchanged then no rendering take place because all renderings are already up to date.



A you can see Remake needs to be able to determine source file of each rendering. And that's achieved by introducing the simple rule for location of rendered files.

<open project directory>

Let's say we have project with configuration file, defining global parameters like resolution, frame rate, etc.

We have some video sequence file (project.blend) and shot file (01/shot.sif).

The rule (agreement) is to put all renderings inside of the “render” subdirectory in the project root under the same path as the source file + format extension.

For example if our default format is PNG, then for “01/shot.sif” the renderpath will be “render/01/shot.sif.png” (that may be a directory if we render to image sequence or just a file if we render to some video file format).

So, as we can see, the rendered footage is separated from source and it's always possible to determine source file from the rendering path.

This rule is very good as it is defines a common structure, a kind of standard. When you working on the project of someone else you don't have to scratch your head “Where can I find this or that?” - you just know where the main things are located.

So this is a very good agreement.

And it's very easy to follow it – just use Remake everytime you need to render something in your project. Remake will automatically put your rendering in proper place according to this rule.

Simple practical example here

For example, let's suppose you want to insert shot.sif into video sequence. Let's render it with Remake OK, done, now we have corresponding directory in the “render” subir. Let's go to the video sequence file and insert rendered footage there. Done. Now rendering of shot.sif is inserted into video sequence and at the same time that means that shot01.sif is linked to video sequence file.

Now let's change shot.sif file by adding colorchart file there. Ok, finished. We also might change other project files. But the funny thing is that to keep our rendering up to date we don't need to render each changed file manually, we don't even need to know which files were changed. The shot01.sif file is already linked to video sequence file, so let's ask remake to render all its dependencies. OK. done and we can see our sequence up to date.

Now let's try to call rendering dependencies again, without changing anything. You can see that Remake analyzed files but no rendering took place – because nothing was changed since the last rendering.

Let's change colorchart now. Re-render video-sequence deps again... You see – sequence correctly reflects all the latest changes.

Finally, suppose we just transferred our source files to another person. To simulate this situation I just kill render directory. Ok, we have no footage now. But it's easy to fix that, isn't it? That's right – render dependencies again. See – all footage restored.

Also we can do normal rendering for our sequence file and we will get finished video file. Simple!

Advantages

- Common structure for animation projects
- Rendering animation from sources is simple as one-click
- Reduced time for re-rendering

Let's summarize the advantages of Remake usage.

Usage cases

- Amazing Sentence

  http://morevnaproject.org/2011/06/19/amazing-sentence/
- The Adventures of Boris Munchausen

  http://munchausenproject.wordpress.com/
- Persona Tomatos - commercial

  http://www.youtube.com/watch?v=5WZPR5h1XwE
- Morevna Project

  http://morevnaproject.org/

The Remake is already used in several projects - open-source and commercial ones. Here are some of them that you can find on the Web.

Development perspectives

- Network rendering
- Templates system
- Transparent integration with software
- GUI

I want to finish my talk by outlining the future development perspectives of Remake.
First one is network rendering – we want to be able to implement parallel rendering on several computers.
Another one is template system. For example adding an empty blender scene what will be automatically adapted to your project settings – resolution, etc. Or more complex template for stereo 3D with a stereo camera rig. There are many templates for Synfig. It would be nice to collect them and be able to insert in the project at user command.
Next important one is a transparent integration with software. Imagine blender working with Remake directly. I.e. you adding sif file into your video sequence, Blender determines that he can't work with .sif format directly, but he detects remake and see that remake can process sif format. Then Blender calls remake, renders sif file and inserts rendered sequence. From the user point of view that's all done transparently, just by clicking on the sif ile.
And finally it would be good to have some GUI to call Remake commands and control project files.

Thank You

http://github.com/morevnaproject/remake

That's all. Thank you for listening.