

Open Source Physically Based Rendering with appleseed



François Beaune
Project Founder

FMX2015

May 05-08, Stuttgart

- Hi everyone!
- Thanks for attending my talk, appreciated! I hope you'll enjoy it.
- My name is François Beaune,
- I'm the founder of the appleseed project.



Fetch

appleseed

- My talk is split in two parts:
 - In the first part, I'll explain what appleseed is and how it works in practice
 - In the second part, I'll talk about Fetch, a short film we completed last year.
- We should normally have time for questions at the end.



appleseed

- Alright, so let's get started with appleseed.

appleseed

- Open source rendering engine
- Designed for **VFX** and **animation**
- Targeted at individuals and small studios

- appleseed is an open source rendering engine
- Specifically designed for visual effects and animation
- That means:
 - Supporting large scenes
 - Lots of geometry
 - Lots of textures
 - Motion blur everywhere
 - Avoiding flicker at all costs
 - Nothing worse than debugging flickering in a heavy shot
 - Being reliable and flexible
- Mainly targeted at individuals and small studios

appleseed

- Started in June 2009
- Small, professional team
- Not our main job

- Started in 2009
 - As a reference: Cycles, the other animation- and VFX-oriented open source renderer, was started in 2011 (as far as I know)
- We all work (or worked at some point) in the industry
- We're doing this in our free time
 - That's the kind of things we like to do
 - Great excuse for many side projects such as animation shorts
 - Allows us to travel quite a bit
 - And to meet many many interesting people
 - So it's a pretty cool hobby really

appleseed

- Pure CPU renderer
- Unidirectional path tracing
- Physically-based
- Highly programmable

- So, at this time, appleseed is a pure CPU renderer.
- GPU rendering is certainly a thing, and *might* be faster
 - But there are many things that we want to do that cannot run on the GPU (today)
 - Still many graphics drivers / CUDA / OpenCL issues and other incompatibility problems
- appleseed is mainly a unidirectional path tracer (like Solid Angle's Arnold)
 - But it has other experimental engines such as light tracing or SPPM
 - We plan to implement and evaluate BDPT and VCM (we already have all the building blocks)
- Physically-based lighting, shading, and cameras
- Highly programmable (we'll come back to this)

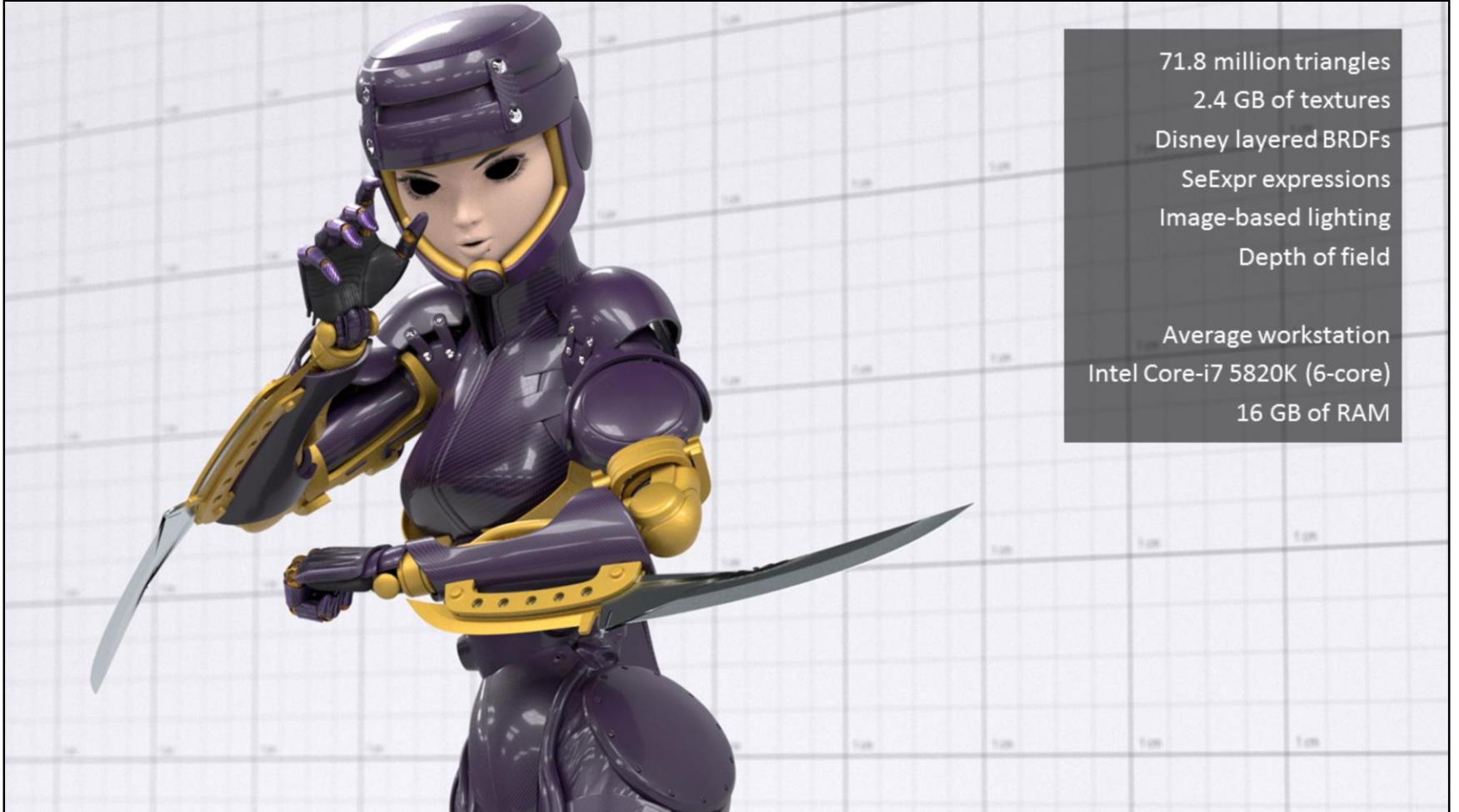
appleseed

LIGHT TRANSPORT	Directional/parallel light Mesh light Purely diffuse emission profile Cone-shaped emission profile Image-based lighting Latitude-longitude environment maps Mirror-ball environment maps Preetham physically-based day sky Hosek & Wilkie physically-based day sky Physically-based sun	MOTION BLUR	Gaffer integration Maya integration Blender integration
RENDERING MODES	Multi-pass rendering Progressive rendering Interactive rendering Scene editing during rendering Spectral rendering (31 bands) RGB rendering Automatic spectral / RGB switching	PRODUCTION FEATURES	Fully open source, MIT license Very clean code CMake build system Full featured C++ API Full featured Python 2.x/3.x API More than 1200 built-in unit tests Hundreds of built-in performance tests Rich, automatic functional test suite
CAMERA MODELS	Pinhole camera Spherical camera Thin lens camera (depth of field) Polygonal diaphragm shapes Image-based diaphragm shapes	REFLECTION MODELS	HACKABILITY
LIGHT SOURCE MODELS	Point light Spot light Gobos	Lambertian BRDF (purely diffuse) Specular BRDF (perfect mirror) Specular BTDF (clear glass) Oren-Nayar Microfacet BRDF Ward Microfacet BRDF Blinn Microfacet BRDF GGX Microfacet BRDF Microfacet BTDF (rough glass) Anisotropic Ashikhmin-Shirley BRDF Kelemen BRDF Disney's Layered BRDF Arbitrary mixture of BRDFs	Fully open source, MIT license Very clean code CMake build system Full featured C++ API Full featured Python 2.x/3.x API More than 1200 built-in unit tests Hundreds of built-in performance tests Rich, automatic functional test suite
		INTEROPERABILITY	PERFORMANCE
		Windows, Linux and OS X (64-bit) OBJ, Alembic, BinaryMesh (proprietary) OpenEXR, PNG OSL shaders	Multithreaded, scalable SSE / SSE2 vectorization Memory-bounded texture cache Multiple Importance Sampling Efficient handling of alpha maps
			TOOLS
			Graphical tool for scene edition Command line renderer Dropbox-based renderfarm tools OSL compiler and tools

- I'm certainly not going to go over all the features available in appleseed
- As you can see there are many
- A lot of them are typical features anyone would expect from a production renderer
- The full list of features is available on our website (URL at the end)



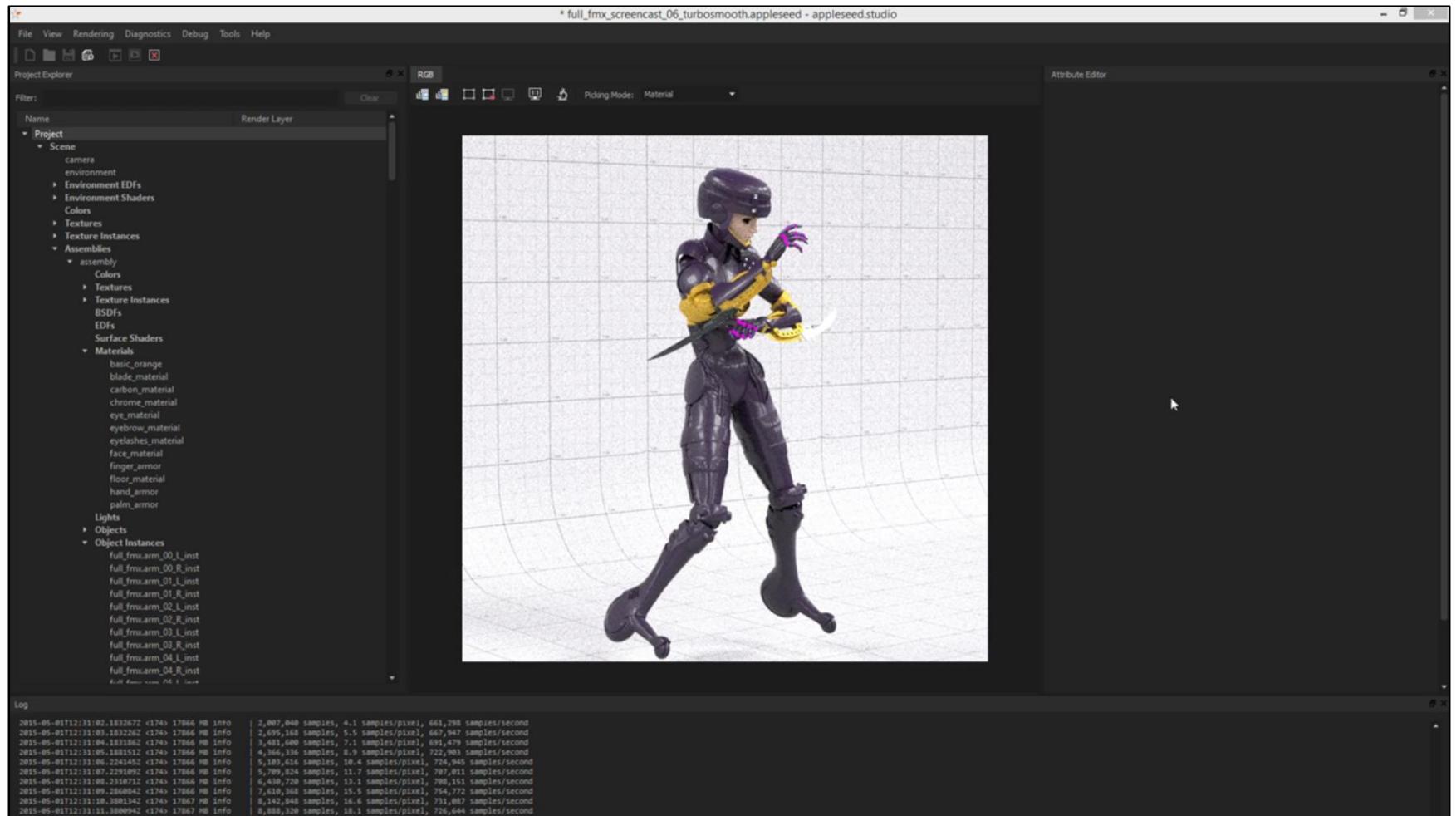
- So, instead, I'd rather show you appleseed in action.
- Here is a model made by my good friend François Gilliot.
- It's a robot girl called Gally, from the manga 'Battle Angel Alita' (Gunnm in Japan, pronounced Ganmu)
- Pardon the missing eyes and highly incomplete lookdev, the model is far from finished.



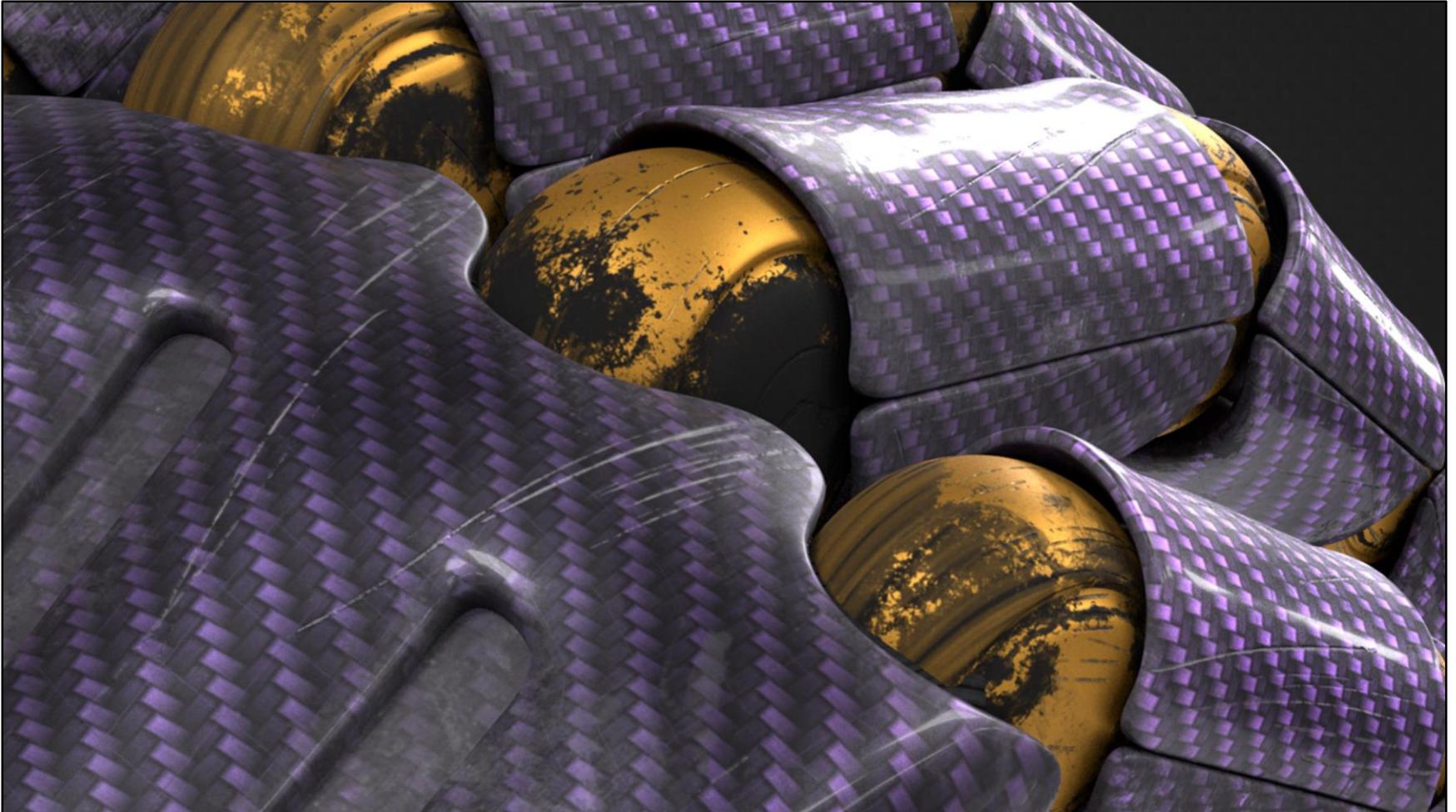
71.8 million triangles
2.4 GB of textures
Disney layered BRDFs
SeExpr expressions
Image-based lighting
Depth of field

Average workstation
Intel Core-i7 5820K (6-core)
16 GB of RAM

- So it's a reasonably large model.



(Video)



- Here is a converged closeup of the finger joints.



- And here is another render of the hand, from a different angle.

appleseed

- Modern
 - Interactive
 - Single pass
 - Tessellation-free
 - Flicker-free

- I want to stress that we're taking a modern approach to rendering
 - In fact most renderers are moving in that direction these days
- We want to have a continuum between interactive rendering and final frame
 - Same rendering engine, same settings, just different quality levels
- We're targeting single pass rendering
 - In particular no prior baking of point clouds or brick maps, no shadow maps...
- As far as possible, we're doing direct ray tracing without pre-tessellating surfaces into triangles, or curves into segments
- Again, as far as possible we're using flicker-free techniques, we're avoiding biased techniques such as all forms of particle tracing

appleseed

- Reliable
 - Avoid (bad) surprises
 - Avoid crashes
 - Avoid regressions
 - Value correctness
 - Incremental change = incremental effect

- We're trying to make appleseed as reliable as we can. That means:
 - Avoiding bad surprises
 - Good in previz = good in final render
 - Avoiding crashes
 - Avoiding regressions.
- We strongly value correctness
 - Different algorithms must converge to the same image
 - Forward path tracing vs. backward path tracing
 - Path tracing vs. particle methods
 - We regularly do these checks, and they are part of our test suite

appleseed

- Flexible
 - Avoid arbitrary limitations
 - Provide tons of public extension points
 - Maximize programmability
 - OpenShadingLanguage
 - Disney's SeExpr
 - Full C++ API
 - Full Python 2.x / 3.x API

- We're also committed to flexibility.
- Obviously we avoid introducing arbitrary limitations, and there aren't many (that I'm aware of)
- We provide tons of extension points
 - It's only a few lines of code to replace a component, or to bypass an entire part of the renderer
- We make sure to provide as much programmability as possible
 - We fully support OSL for shading
 - We also support SeExpr expressions in a growing number of places
 - We have full C++ and Python APIs

appleseed

- Hackable
 - Fully open source
 - Liberal license (MIT) from the start
 - Everything hosted on GitHub
 - Development fully in the open
 - Using only open source or free tools
 - Welcoming, helpful, mature community

- Hackability means removing barriers to entry.
- Everything we do is 100% open source
- All our code is under the MIT license, from the beginning
 - That means commercial embedding is OK
- Everything is hosted on GitHub
 - Source code for appleseed and all related tools
 - Issue tracker
 - Documentation
 - Wiki
 - Web site...
- We're only using and relying on open source software
 - Except Visual Studio on Windows (which is free but not open source)

Team & Process

- I want to quickly highlight the role and contributions of our team members.



François Beaune



Esteban Tovagliari



François Gilliot



Jonathan Topf



Hans Hoogenboom



Joel Daniels



Dorian Fevrier



Haggi Krey



Srinath Ravichandran



Marius Avram

- So here are the principal members of the team.

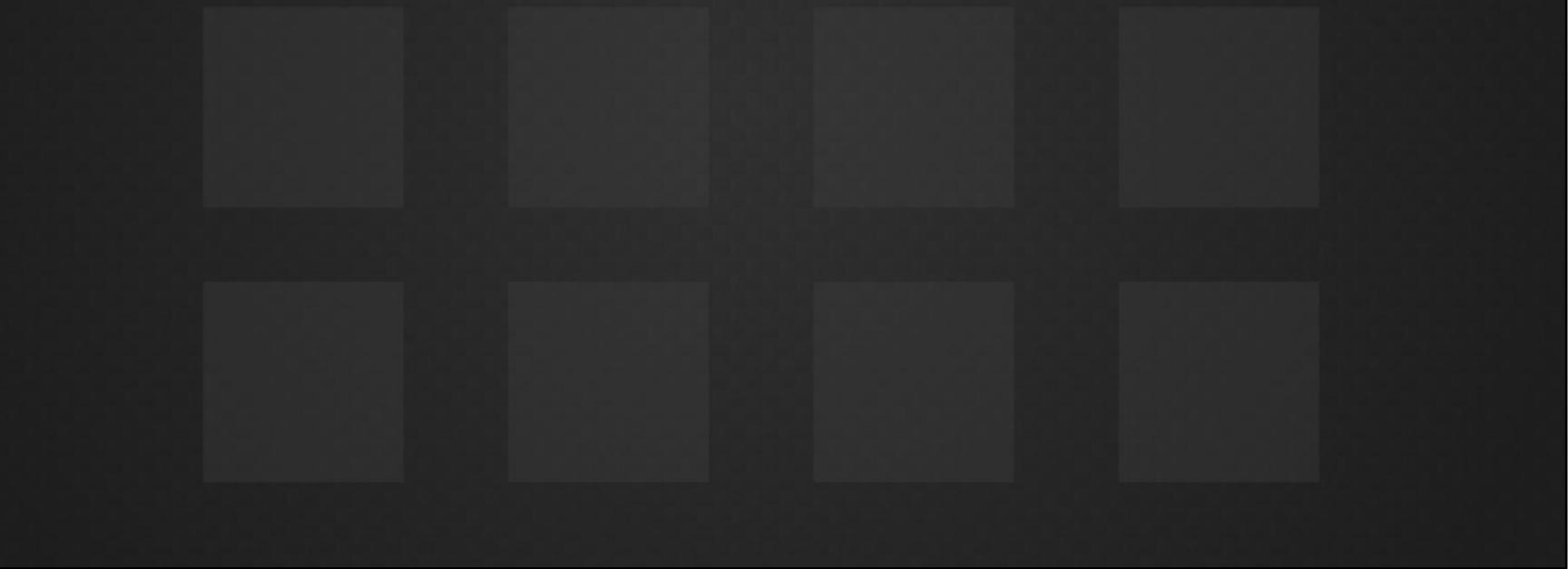
R&D



François Beaune



Esteban Tovagliari



- At the moment we're only two developers working on the core renderer.

GSOC '14 Students



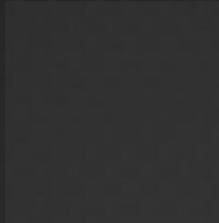
Srinath Ravichandran



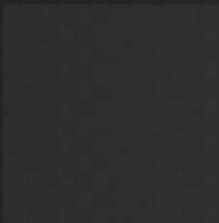
Marius Avram

- We were fortunate enough to participate to Google Summer of Code last year,
- And we had two very talented students that did a really good job
 - One who worked on curve rendering for hair & fur
 - One who worked on the material editor in appleseed.studio

Exporters & Integrations



Esteban Tovagliari



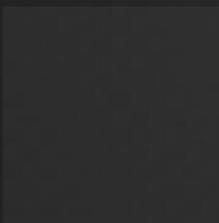
Jonathan Topf



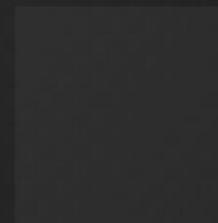
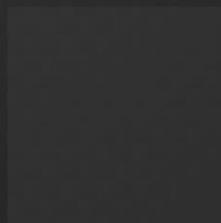
Hans Hoogenboom



Joel Daniels



Haggi Krey



- These guys do a great job at connecting appleseed with DCC apps such as Maya, Blender or Gaffer.

Fetch



François Beaune



François Gilliot



Jonathan Topf

- And finally this is the team that worked on the short film Fetch about which I'll talk next.

appleseed

- Core practices and values
 - Collective code ownership
 - Continuous refactoring
 - Pull requests reviews
 - Unit tests
 - End-to-end tests
 - Performance regression tests

- Finally, I want to quickly mention a set of core values and practices that we share, since these have a direct impact on the quality of our work:
 - Collective code ownership: We're all allowed to touch or improve all of the code
 - Continuous refactoring: We keep simplifying and clarifying the code whenever we can
 - We do friendly but honest reviews of pull requests
 - PRs are usually good for merging after a couple rounds of reviews and fixes
- We do lots of testing, most of it is automated
 - This allows us to refactor the code while avoiding regressions

appleseed

Selected Works

- Here are a few recent works done with appleseed.



Light & Dark (BBC Four Documentary)

- Here's a short clip from Light & Dark (video).



Light & Dark (BBC Four Documentary)

- This is a frame from a CG sequence in Light & Dark, one of two documentaries that were made for BBC Four and that aired last year on British TV.



Light & Dark (BBC Four Documentary)

- Another one.

Character designs by appleseed users





Fetch, a very short film

- This is a frame from Fetch, the short film we'll talk about next.



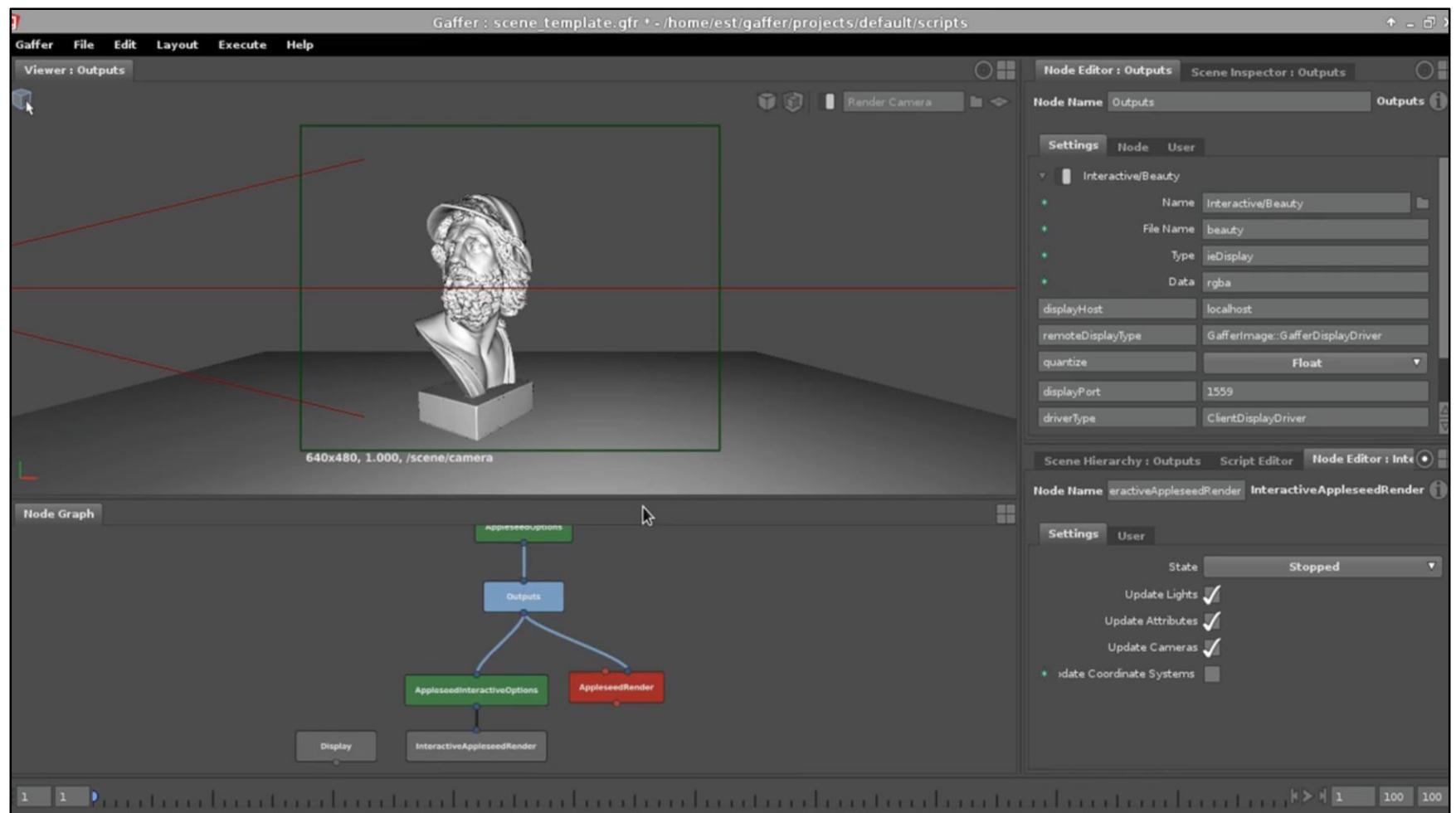
Fetch, a very short film

- And another one.

appleseed

appleseed now fully integrated into Image Engine's Gaffer

- appleseed is now fully integrated into Gaffer
 - Gaffer is an open source lookdev application by Image Engine
 - Which is a VFX company based in Vancouver, which worked on Elysium, Zero Dark Thirty, District 9...
- This is the result of the phenomenal work by Esteban Tovagliari, in conjunction with John Haddon from Image Engine



- Here's a quick demo of appleseed inside Gaffer.

appleseed

Welcoming contributions!

- We're welcoming contributions of all kinds!
- So if you feel like writing some code, or doing testing, get in touch with us!

appleseed

Home

<http://appleseedhq.net/>

GitHub

<https://github.com/appleseedhq/appleseed>

Development Mailing List

<https://groups.google.com/forum/#!forum/appleseed-dev>

Twitter

<https://twitter.com/appleseedhq>

- I put the principal links on this slide, but you can also just type 'appleseed renderer' in Google and you should be good to go.



Making Fetch

- Alright, let's move to the second part of this talk: Making the short film Fetch.

Making Fetch

- Initiated “Project Mescaline” in June 2012
- Goals:
 - Test & validate appleseed on a small production
 - Showcase & promote appleseed
 - Sharpen our skills
 - Have fun with friends
- Constraints:
 - Final render 100% appleseed
 - Tiny budget

- We initiated what we called ‘Project Mescaline’ (I don’t exactly remember why) in June of 2012.
- The main goal of this project was to test and validate appleseed on a small production
- We also wanted to have some cool material to showcase and promote appleseed
- It was also a good occasion to sharpen our skills, and have fun with friends (which we totally did)
- We had two main constraints though:
 - The final render had to be 100% done with appleseed
 - And we only had a tiny budget.

Making Fetch

- Small team:
 - 1 for direction & art
 - 1 for pipeline & render
 - 1 for sound effects & soundtrack (late in project)
 - Help from friends
- Strictly free-time / rainy days project
- Effort:
 - Planned: 8 months
 - Actual: 19 months ☺

- As I showed earlier, we were a very small team:
 - One person (François Gilliot) was responsible for the direction, and for all the graphics arts
 - One person (me) was responsible for pipeline setup and the final render
 - And one person (Ulric Santore) was responsible for sound effects and soundtrack
 - He was only involved at the end of the project, and he did a terrific job
 - We also got the occasional help from friends, in particular Jonathan Topf for the Maya-to-appleseed exporter
- Like appleseed, this was a strictly free-time / rainy days project.
- We kind of blew the schedule... But it's OK ☺

Making Fetch

- “Fetch, a very short film”
- 2 minutes hand-animated short
- Targeted at kids
- Miniature look
- Fully rendered with appleseed

- So the film is appropriately called ‘Fetch, a very short film’
- It’s a 2-minute hand-animated short
- Targeted at kids
- We went for a miniature look
- Definitely inspired by the animated film Coraline, produced by Laika
- And of course, as this was the goal, every single pixel was rendered by appleseed



(Video)

Making Fetch

- Pipeline
- Render Setup
- Render Farm
- Conclusion

- So I'll be talking about three technical aspects of the making of Fetch
 - Our render pipeline
 - How we did the render setup
 - And our custom render farm

Making Fetch

Pipeline

Making Fetch – Pipeline

- Modeling, animation, lookdev in 3ds Max
 - Tool of choice for the artist
- Lookdev mostly with V-Ray
 - Integrated in 3ds Max

- All modeling, animation and lookdev was done in 3ds Max
 - There wasn't much discussion about it, it was just the tool of choice of the artist.
- Lookdev was mostly done with V-Ray
 - Again because it's the tool of choice of the artist
 - Also because the integration of V-Ray in 3ds Max is solid

Making Fetch – Pipeline

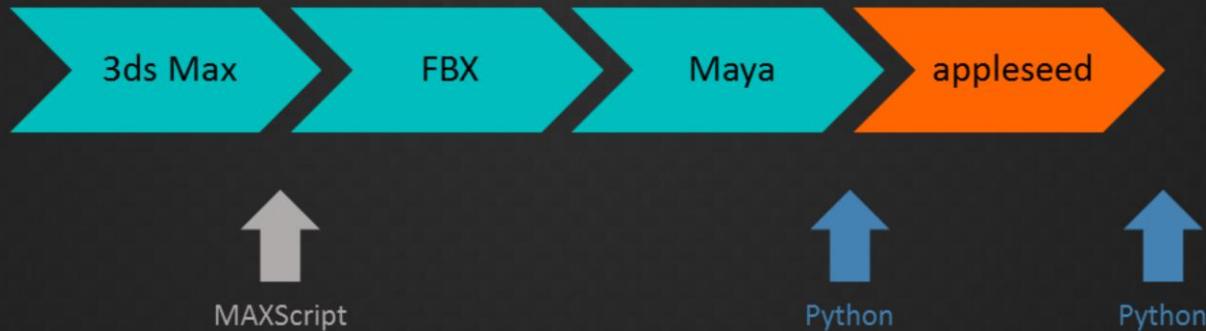
- Problem: no 3ds Max-to-appleseed exporter
- Writing a full-featured exporter for 3ds Max too big of a project
- Solution:



- The first problem was of course that, while we had an exporter for Maya (called mayaseed), we didn't have one for 3ds Max.
- We thought we wouldn't have time to write a full-featured exporter for 3ds Max
 - On a side note, it turned out we would have had time, and maybe that would have been a wise decision...
- Our 'brilliant' solution was to rely on Maya for the export, and on the FBX file format for scene data transport...

Making Fetch – Pipeline

- Problem: no 3ds Max-to-appleseed exporter
- Writing a full-featured exporter for 3ds Max too big of a project
- Solution:



- Of course, we had to automate a bunch of intermediary tweaks to get it all to work:
 - In 3ds Max, before the FBX export
 - In Maya, right after the FBX import
 - And right after the export to appleseed scene files

Making Fetch – Pipeline

- FBX format would lose lots of information
 - Area lights
 - Gobos
 - DOF parameters...
- Several custom scripts to remedy this
 - 3ds Max side (MAXScript)
 - Store various info into custom attributes
 - Prepare the scene before FBX export
 - Maya side (Python)
 - Retrieve info from custom attributes
 - Adjust materials



- One of the reason was that the FBX file format is totally not suited for transporting film scenes across DCC apps
 - It cannot adequately represent
 - Area lights
 - Gobos (projection textures on spot lights)
 - Depth of field parameters...
- So the first set of scripts were ran in 3ds Max, before FBX export
 - They would store as custom attributes everything that cannot be represented by FBX
 - And generally speaking, they would prepare the set before export
 - You can see here on the right the UI we built for these scripts
- The second set of scripts were ran in Maya, after FBX import
 - Retrieve the info from custom attributes and apply them to the scene
 - Adjust materials somewhat

Making Fetch – Pipeline

- Initial lookdev mostly with V-Ray 3
- Materials translated to appleseed
 - Automatic translation during export
 - Lots of post-export tweaks
 - Automatic tweaks via Python scripts

- As I explained earlier, the lookdev was done with V-Ray 3
- That meant we had to translate V-Ray materials to appleseed
 - That was mostly done automatically with our pre-FBX and post-FBX scripts
 - But some more tweaking was necessary after the export to appleseed
 - A Python script that would directly alter the appleseed scene files (looking for objects and materials by name)



- Intermezzo: this is the color script of Fetch

Making Fetch

Render Setup

- Let's talk now about our render setup...

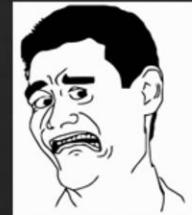
Making Fetch – Render Setup

- Art direction called for:
 - Miniature look = realistic lighting + shallow DOF
 - Mostly forest shots with almost no direct illumination
 - Millions of grass blades and tree leaves in nearly every shot
 - All translucent (thin translucency)
 - All using alpha cutouts
 - Image-based lighting in 25% of the shots
 - Many scenes with really strong motion
 - Transformation and deformation

- Usually, stop motion look means no motion blur
 - But we wanted to demonstrate appleseed's motion blur capabilities so we decided to use it nevertheless

Making Fetch – Render Setup

- Art direction called for:
 - Miniature look = realistic lighting + shallow DOF
 - Mostly forest shots with almost no direct illumination
 - Millions of grass blades and tree leaves in nearly every shot
 - All translucent (thin translucency)
 - All using alpha cutouts
 - Image-based lighting in 25% of the shots
 - Many scenes with really strong motion
 - Transformation and deformation



Making Fetch – Render Setup

- Physically-based materials & lighting
- Unidirectional path tracing, 2 bounces
- 64-400 samples/pixel depending on DOF and MB
- Single pass, no baking whatsoever
- One AOV per light (4-6 lights per shot)
- Plus a few special AOVs
 - Girl's hair
 - Wolf's eyes...

- So this is how we setup our rendering:
 - Of course we went for physically-based materials and lighting
 - Since that's what would allow us to achieve a miniature look
 - We limited path tracing to two bounces
 - Not sure more bounces would have been much slower since most surfaces are rather dark
 - We used anywhere from 64 to 400 samples/pixel depending on DOF and MB

Making Fetch – Render Setup

- Full HD resolution (1920x1080)
- 24 frames/second
- 2767 frames (~ 115 seconds)

- We chose to render at full HD resolution
- And we chose 24 frames/second
 - In hindsight, we could have chosen a lower frame rate, which would have made sense in the stop motion context

Making Fetch – Render Setup

- 3120 individual scenes to render
 - 2767 frames + a couple backgrounds rendered separately
- 32 GB of final render data
 - OpenEXR textures (RLE-compressed)
 - Proprietary geometry format (LZ4-compressed)
- Tens of thousands of files



- Intermezzo: just a drawing

Making Fetch

Render Farm

- Alright, so let's talk now about how we actually managed to render that short film...

Making Fetch – Render Farm

- Obviously too much work for one or even a couple machines
- No money meant:
 - Not buying additional machines
 - Not renting a render farm
 - Not paying for Amazon Web Services
- So?

- Obviously we had way too many frames to render for a single machine, or even a couple of machines
- And since we had no money to spare, we couldn't
 - Buy additional machines to build our own render farm
 - Rent a render farm
 - Build a farm using AWS

Making Fetch – Render Farm

- Friends to the rescue!
- Challenges:
 - 32 shots, tens of thousands of files, GB of data
 - Friends all around the place in Europe
 - Random machines
 - Random OS
 - Machines only available occasionally
 - Many machines behind firewall / NAT
 - No technical expertise or rendering experience for most of them

- At this point we decided to involve our friends.
- But that brought its own set of challenges.

Making Fetch – Render Farm

Solution:

DYI render farm based on Dropbox

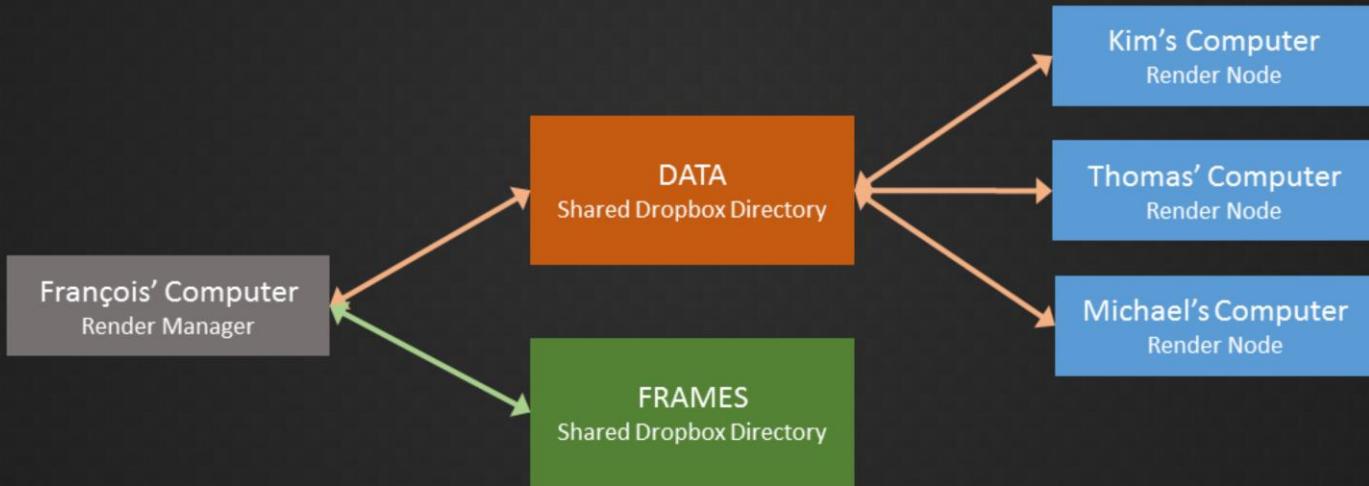
- The solution to this nightmare came in the form of a custom render farm system built around Dropbox
- The inspiration for this came from a tiny script that one of our team member, Jonathan Topf, who wrote to render the animations from the Light & Dark documentaries I talked about earlier.

Making Fetch – Render Farm

Use Dropbox as **delivery channel**,
and for **command & control**

- The core idea is to use the Dropbox shared directory mechanism:
 - To deploy appleseed binaries to render node
 - Reliably send scene data to render nodes
 - Reliably send rendered frames back to some kind of ‘master’ node

Making Fetch – Render Farm



- So this is the overall architecture of our system.

Making Fetch – Render Farm



- Shared directory
- Assume Dropbox Basic accounts (free!) = 2 GB
- Hosts:
 - appleseed binaries for Windows, Linux and OS X
 - Data for one or multiple partial shots

- The central piece is a shared directory we call DATA
 - Free Dropbox Basic accounts are limited to 2 GB, so that's the upper limit for the content of this directory.
 - In this directory we'll store three things:
 - appleseed binaries for Windows, Linux and OS X
 - Shot data (scene files, textures, geometry, etc.)
 - A few rendered frames
 - This directory **is** the central delivery and command & control mechanism.
- It's important to note that, regarding shot data, we may have:
 - multiple different shots at the same time
 - partial shots (only parts of the frames)

Making Fetch – Render Farm

- Shared directory on Dropbox Pro accounts
- Hosts all rendered frames
 - Ended up with 140 GB worth of OpenEXR files
- Only shared between team members



- We also have a second directory we call FRAMES
 - For this one we assume Dropbox Pro accounts, so limited to 1 TB of data
 - This directory hosts all the frames renderer so far
 - We ended up with about 140 GB worth of OpenEXR files
 - This directory is only shared with team members, not with the render farmers

Making Fetch – Render Farm

- A variety of 64-bit machines
 - Windows Vista, 7, 8
 - Linux
 - OS X
- Mostly quad core machines
- Typically available nights and week-ends
- Render nodes run the render node script
- Users free to kill render node script at any time

Kim's Computer
Render Node

Thomas' Computer
Render Node

Michael's Computer
Render Node

- Then we have the Render Nodes themselves
 - These are just random 64-bit machines running a variety of operating systems
 - Windows Vista, Windows 7, Windows 8
 - Linux
 - OS X
 - We had mostly quad core machines, but not only
 - These machines were typically available on nights and week-ends only
 - Render nodes run a Python script we call the render node script
 - Acquires render jobs and executes them
 - Machine owners were free to kill the render node script at any time to reclaim their machine

Making Fetch – Render Farm

- Render nodes run a Python script:

Loop:

“Acquire” scene by appending a per-machine suffix to scene file

Render scene

Move rendered frame files to “frames” subdirectory in **DATA**

Move rendered scene file to “archive” subdirectory in **DATA**

- The render node script runs a main loop:
 - It first ‘acquire’ a **random** available scene file by appending a per-machine suffix (a short id) to the file
 - It then render the scene locally
 - Once done, it moves the rendered frame (up to a dozen OpenEXR files) to a ‘frames’ subdirectory
 - And finally it moves the scene file to an ‘archive’ subdirectory

Making Fetch – Render Farm

François' Computer
Render Manager

- Underpowered Core i5 laptop
- Managing rendering:
 - Upload/remove shot data as required
 - Honor 2 GB size limitation of DATA at all times
 - Move rendered frames from DATA to FRAMES
 - Monitor and print render farm health, activity and progress
- Running 24/7

- Finally we have the Render Manager, which was just my laptop (my main machine at the time)
 - Definitely a low-spec machine, certainly too weak for any serious compute task
 - But enough to manage rendering (and to develop appleseed)
 - So the tasks of the Render Manager are:
 - Upload shot data as required
 - Remove unused shot data to honor the 2 GB size limitation at all times
 - Move rendered frames from DATA to FRAMES
 - Monitor and print the render farm health, activity and progress
- My machine was running nearly 24/7 at the time
 - I actually surprised it didn't kill it

```

Term --max-size 1536 --source . --target "c:\franz\Dropbox\Render Farm 1\data" --frames "c:\franz\Dropbox\Project Mescaline\final frames\36_28_00"
2014-01-16 18:34:22.405000 mgr info --- starting logging ---
2014-01-16 18:34:22.406000 mgr info running rendermanager.py version 2.4.
2014-01-16 18:34:22.937000 mgr info gathering files...
2014-01-16 18:34:22.946000 mgr info found 280 source files in .
2014-01-16 18:34:22.946000 mgr info found 25 completed files (all shots) in c:\franz\Dropbox\Render Farm 1\data\_archives
2014-01-16 18:34:22.947000 mgr info found 8 in-progress files (all shots) in c:\franz\Dropbox\Render Farm 1\data
2014-01-16 18:34:22.947000 mgr info found 67 uploaded files (all shots) in c:\franz\Dropbox\Render Farm 1\data
2014-01-16 18:34:22.947000 mgr info -----
2014-01-16 18:34:22.949000 mgr info PROGRESS: 25/280 completed (8.93 %), 8 rendering, 247 pending
2014-01-16 18:34:22.951000 mgr info frame assignments:
2014-01-16 18:34:22.951000 mgr info 36_28_00.0058.appleseed: ku
2014-01-16 18:34:22.951000 mgr info 36_28_00.0071.appleseed: yd_daesign
2014-01-16 18:34:22.952000 mgr info 36_28_00.0010.appleseed: ta
2014-01-16 18:34:22.953000 mgr info 36_28_00.0003.appleseed: fg_daesign
2014-01-16 18:34:22.953000 mgr info 36_28_00.0037.appleseed: nb_daesign
2014-01-16 18:34:22.953000 mgr info 36_28_00.0030.appleseed: mjp
2014-01-16 18:34:22.953000 mgr info 36_28_00.0076.appleseed: yc_daesign
2014-01-16 18:34:22.954000 mgr info 36_28_00.0013.appleseed: rdc
2014-01-16 18:34:22.954000 mgr info pings:
2014-01-16 18:34:22.958000 mgr info yd_daesign: 0 h 23 m 48 s ago (at 2014-01-16 18:10:34.011000)
2014-01-16 18:34:22.959000 mgr info nb_daesign: 0 h 30 m 51 s ago (at 2014-01-16 18:03:31.232000)
2014-01-16 18:34:22.960000 mgr info fg_daesign: 0 h 14 m 17 s ago (at 2014-01-16 18:20:05.300000)
2014-01-16 18:34:22.961000 mgr info yc_daesign: 0 h 20 m 35 s ago (at 2014-01-16 18:13:47.921000)
2014-01-16 18:34:22.961000 mgr info mjp: 4 h 13 m 44 s ago (at 2014-01-16 14:20:38.446000)
2014-01-16 18:34:22.961000 mgr info ku: 2 h 49 m 26 s ago (at 2014-01-16 15:44:56.644000)
2014-01-16 18:34:22.962000 mgr info sdc: 1 h 49 m 37 s ago (at 2014-01-16 16:44:45.940000)
2014-01-16 18:34:22.962000 mgr info ta: 1 h 32 m 45 s ago (at 2014-01-16 17:01:37.187574)
2014-01-16 18:34:22.962000 mgr info size of target directory: 1550.0000 MB (100.00)
2014-01-16 18:34:22.963000 mgr info moving frames...
2014-01-16 18:34:22.963000 mgr info updating dependency database of uploaded files...
2014-01-16 18:34:23.578000 mgr info added 36_28_00.0001.appleseed
2014-01-16 18:34:24.304000 mgr info added 36_28_00.0002.appleseed
2014-01-16 18:34:25.030000 mgr info added 36_28_00.0003.appleseed
2014-01-16 18:34:25.815000 mgr info added 36_28_00.0006.appleseed
2014-01-16 18:34:26.545000 mgr info added 36_28_00.0007.appleseed
2014-01-16 18:34:27.254000 mgr info added 36_28_00.0008.appleseed

```

- Here is a random screenshot I found while making this presentation
- Nevermind the red rectangle highlighting the machine pings
 - I was probably just excited to show this new feature to someone at the time this screenshot was made
- So in this picture you can see:
 - A summary of what the DATA directory contains
 - Frame files being rendered by machines
 - Machines are identified by a short id, typically the initials of its owner
 - Pings, to check when we last got news from a machine
 - If you're curious, pings were derived from the 'Last Modified' date on the scene files
 - And then what actions the render manager is taking

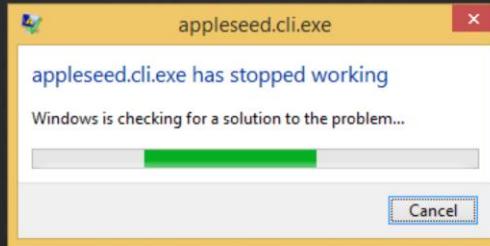
Making Fetch – Render Farm

- Render Manager Robustness
 - “Rendering state” fully implicit
 - Render manager free to start/stop/crash at any time

- The key to making the render manager robust was to make it state-free
 - The ‘rendering state’ was entirely determined by the files in DATA
- That meant that I could start or stop the render manager without impacting rendering
 - For instance to fix a bug in the script
- If the render manager crashes or stops:
 - Render nodes simply run out of work
 - Or the DATA shared directory fills up and nodes no longer gets Dropbox updates

Making Fetch – Render Farm

- Render Nodes Robustness
 - Not all geometry files or textures available to render given scene
 - On Windows: appleseed crash = Windows Error Reporting Message Box



- Geometry files or textures could be missing to render a given frame
 - So the render node scripts has to check if all dependencies are present before rendering
 - Parse scene file (XML) and extract file names
- On Windows, if appleseed crashes, by default a message box opens
 - That stalls the render node until someone gets in front of the computer to manually close the message box
 - So the render node script disables Windows Error Reporting on startup (and restores it on exit)

Making Fetch – Render Farm

- Advantages

- Easy for friends to join & participate
- Reliable transport of scene data and rendered frames
- Easy to add/remove render nodes
- Easy to update new appleseed binaries
- Easy to analyze performance and crashes of render nodes
- Eventually quite robust



- Intermezzo: a close-up of the wolf with his curious feather-like fur...

Conclusion

- Let's conclude this section, and the talk.

Mescaline Render Planning																	
Shot	Version	Description	3ds max start / end	Needs DOF?	Needs Sky?	Shutter Open Duration	Pixel Samples	Light Samples	Env Samples	Pending Remarks	Color Legend:	Ready to Import	Ready to Render	Rendering	Done	Broken	Re-render
00	08	opening shot on the valley	0 132	Y	N	0.5	100	1	-								
01	26	she appears on the hill	0 115	Y	YES	0.5	100	8	16								
03	14	she runs down the hill	0 170	Y	YES	0.25	64	16	1								
04	07	she enters the forest	0 105	Y	N	0.5	64	10	-								
04_bg	07	background	0 105	NO	YES	no motion blur	16	1	1								
05	20	she plays in the forest 1	0 135	Y	N	0.5	100	4	-								
06	29	she plays in the forest 2	0 300	Y	N	0.5	80	4	-								
07	12	she sees the wolf	0 80	Y	N	0.5	200	16	-								
09	18	she waits and walks by the wolf	0 168	NO	N	0.5	200	1	-								
10	19	she walks by the wolf	0 118	Y	N	0.5	64	4	-								
11	24	wolf stands up	0 168	Y	N	0.5	100	4	-								
17	26	she jumps over a large root and wobbling slope	15 130	Y	N	0.5	200	1	-								
26	20	she stops at the edge of the cliff	0 115	Y	YES	0.5	100	1	1								
26_bg	20	background	0 115	NO	N	0.5	4	1	-								
28	12	she looks around to find a way	0 73	Y	YES	0.5	100	1	1								
28_bg	12	background	0 73	NO	N	0.5	4	1	-								
31	15	the girl turns back to the forest	0 50	Y	YES	0.5	100	1	1								
33	07	she puts her hand in the basket	0 61	Y	YES	0.5	100	1	1								
36	28	wolf arrives and wants to play	0 280	Y	YES	0.5	64	1	1								
37	13	she runs towards the exit	0 42	Y	N	0.5	100	1	-								
38	02	closeup wolf face	0 45	Y	N	0.5	100	1	-								
39	00	closeup wolf face	0 47	Y	N	0.5	400	1	-								
40	01	closeup hood feet	0 75	Y	N	0.5	200	1	-								
41	01	closeup wolf face	0 27	Y	N	0.5	400	1	-								
42	01	closeup wolf feet	0 33	Y	N	0.5	200	1	-								
43	05	she turns away and runs	97 148	Y	N	0.5	200	1	-								
44	11	she runs and jumps over a gap	0 40	Y	N	0.5	400	1	-								
45	04	she looks behind while she runs	0 34	Y	N	0.5	200	1	-								
46	05	she sees the exit	0 44	Y	N	0.5	800	1	-								
50	01	wolf runs	0 50	Y	N	0.5	300	1	-								
51	30	wolf runs toward the hood	0 74	Y	N	0.15	200	1	-								
53	19	the girl tries to lift the branch without success	0 60	Y	N	0.25	64	4	-								
Total number of frames			3120														

- It's hard to say how long rendering eventually took
 - Since it was mixed up with lots of activities such as preparing shots, exporting them, etc.
- A couple weeks seems like a reasonable estimate

Making Fetch – Conclusion

- Special developments
 - Efficient handling of massive number of alpha cutouts
 - Dropbox-based render farm tools
 - Vast improvements to Maya-to-appleseed exporter (mayaseed)
- Everything has been released

- Making Fetch led to a few interesting developments:
 - Very efficient handling of large number of alpha cutouts
 - The Dropbox-based render farm system we just talked about
 - And vast improvements to mayaseed, our Maya-to-appleseed translator
- Everything that we did for Fetch finds its way into official releases.

Making Fetch – Conclusion

- appleseed one of the most reliable component of the pipeline
- Did not have to worry about:
 - Flickering
 - Glitches in the middle of a shot
 - Unpredictable catastrophic slowdown

- Retrospectively, appleseed was certainly one of the most (if not the most) reliable component of the pipeline
- We did fix a few important bugs at the very beginning of the project, but after that it was very reliable
- Flickering has never been a concern, and we didn't get any
- Similarly, when we did encounter render glitches in the middle of the shot, they were due to a bad scene setup, not due to appleseed
- We didn't suffer from unpredictable performance problems such as catastrophic slowdown...

Making Fetch – Conclusion

- Only two questions:
 - What render settings?
 - How long will it take?

- We were basically left with only two questions:
 - What render settings to use for acceptable noise levels & render times?
 - How long will the render take for any given shot?

Making Fetch – Conclusion

- What would we do differently today?
 - Export Alembic files from 3ds Max
 - Lookdev in Gaffer
 - Real hair?
 - OSL shaders?

Making Fetch – Conclusion

- Published on Vimeo
- Picked up by many big animation channels, ended up on YouTube
- Great reception on the web
- Some really nice articles written about the project

Making Fetch – Conclusion

- Official TIFF Kids 2015 selection!



- We were actually invited to Toronto last month to present the film!



Thank you!



Questions?

- We've got a few minutes, I'll be happy to answer any question!

Extras

There's never enough!

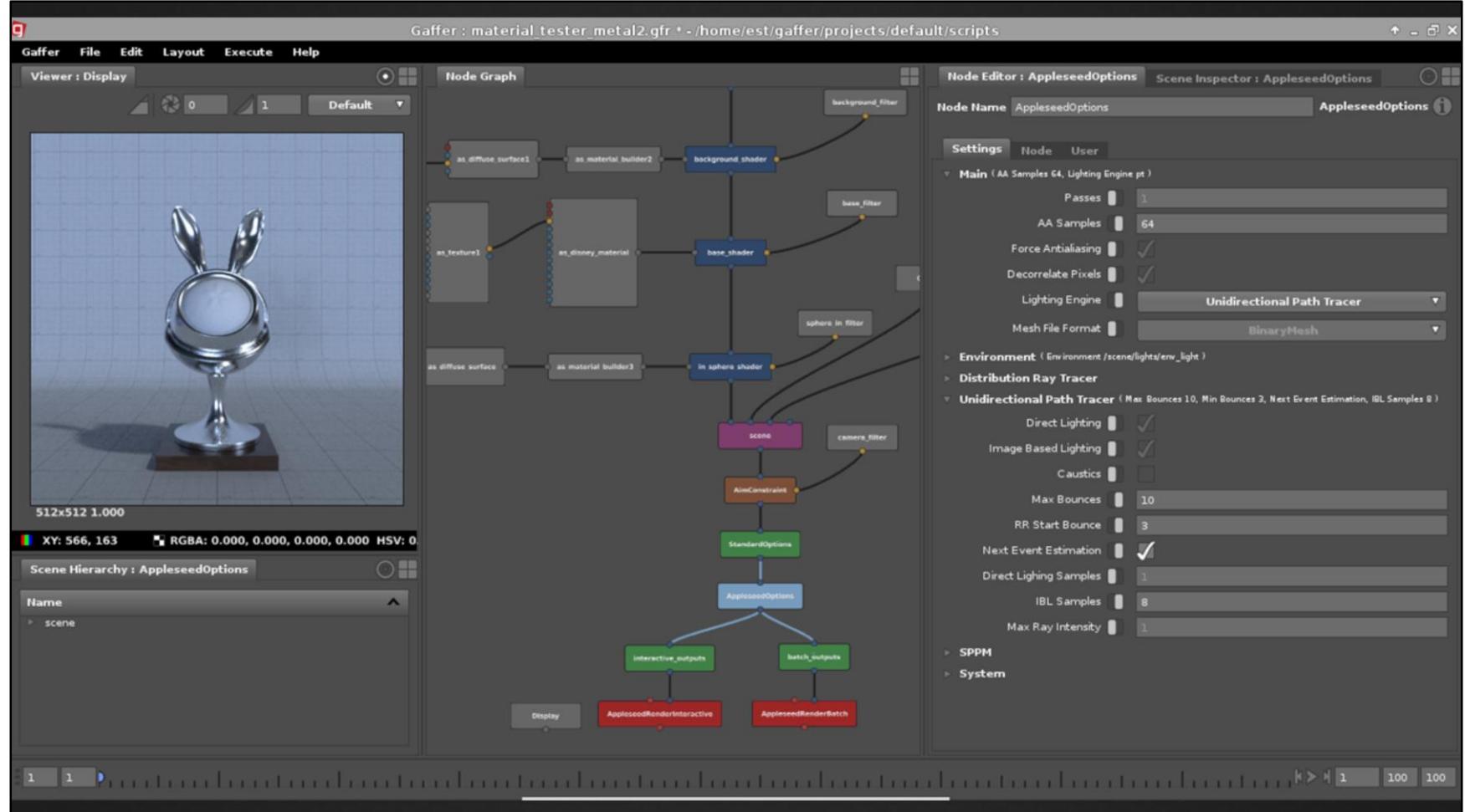
Additional References

Direct Ray Tracing of Full-Featured Subdivision Surfaces with Bezier Clipping
<http://jcgt.org/published/0004/01/04/>

- Direct ray tracing of Catmull-Clark subdivision surfaces by Takahito Tejima et. al. (Pixar)
 - Intersection of Bézier patches without pre-tessellation
 - Supports full set of RenderMan and OpenSubdiv features
 - Hierarchical edits, creases, semi-sharp creases, corners, etc.

appleseed

- Many important features still missing
 - Volume rendering
 - Subsurface scattering
 - Subdivision surfaces
 - Displacement
 - Robust, complete, performant Maya integration
 - Documentation



- Here's a screenshot of appleseed inside Gaffer.