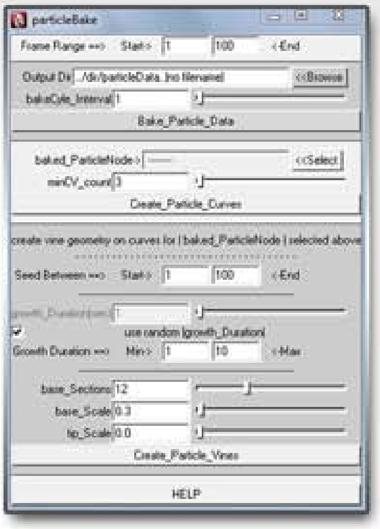
ENTRANCE SCENE



particleBake

Frame Range ==> Start> 1 | 100 <End

Output Dir ../dk/particleData. (no filename) <<Browse

bakeCyle_Interval 1

Bake_Particle_Data

baked_ParticleNode > ----- <<Select

minCV_count 3

Create_Particle_Curves

create vine geometry on curves for | baked_ParticleNode | selected above

Seed Between ==> Start> 1 | 100 <End

growth_Duration(sec) 1

✓ use random |growth_Duration|

Growth Duration ==> Min> 1 | 10 <Max

base_Sections 12

base_Scale 0.3

tip_Scale 0.0

Create_Particle_Vines

HELP

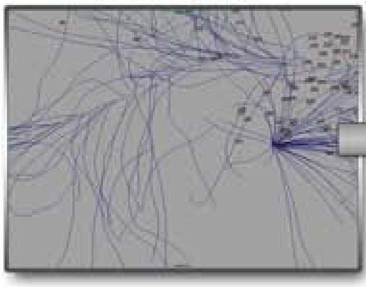
The vine like geometry in the entrance scene is created and animated with a MEL script I wrote. The script bakes out particle data into '.txt' files which consist of the world position vectors of every particle during their lifespan. Afterwards, these baked out transformation data is used to create curves in the scene. NurbsCircles will be attached to the bases of each created curve and they will be aimed to the appropriate direction using constraints. And finally, nurbs vine geometry will be created by extruding the circles along curves.

'bakeCycleInterval' option is for exporting particle data within the declared frame cycle from this slider. A higher frame cycle can be used for baking out heavy particle simulations for a faster performance. A low frame cycle would be more accurate but the process will be slower.
'minCV_count' slider is a useful option for getting rid of undesired short curves that will be created by using short aged particles' data. Any curve that has a lower CV count than this value will not be created. Additionally, the 'base & tip scale' and the sections of the nurbs vine geometry can be specified from the interface.

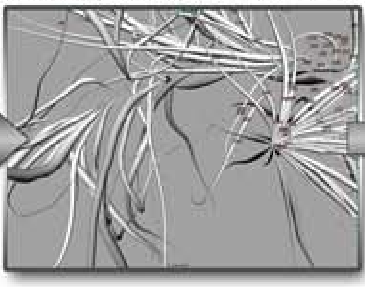
The script also has the ability to animate the vines. The script will utilise a vine growth animation automatically where a vine will start growing from the base of a curve and stop at the tip of it. Timing of the animation like growth duration, start/end frame or random seeding can be controlled from the GUI.

In this project, I used custom RenderMan shaders I wrote in RSL for the final render by assigning a custom noise displacement and a rim surface shader to the vine geometry. Finally I used After Effetcs just for color corrections and glow effects.

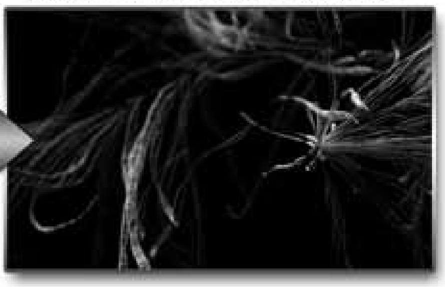
curve creation



geometry creation + animation



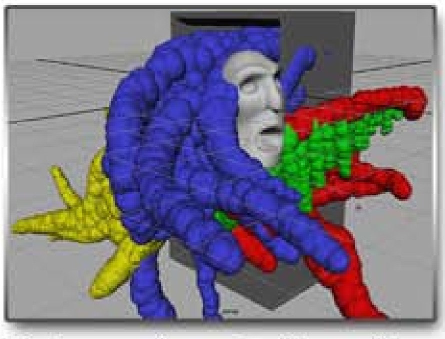
custom RMan Shaders applied



Work Done:
Animation, Scripting, Shader Writing, Rendering

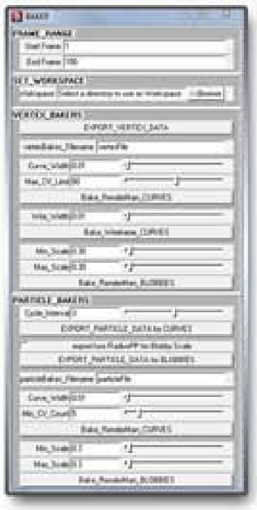
Software used:
Maya/RenderMan, Adobe After Effects

ICICLE MAN



In the scene, the icicles coming out of the refrigerator are formed with a particle simulation. In the simulation, I have 4 different particle emitters as colored above and 8 particle nodes within. The face is also added to the animation as a mesh, not as particles.

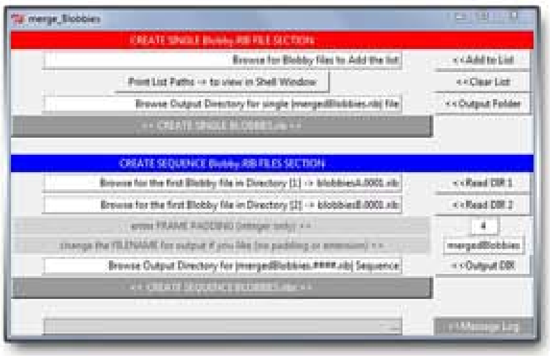
For rendering this simulation as a whole blobby object, all data from these seperate particle nodes and the mesh face should be merged into a single file. To do this, I used a custom MEL script I wrote.



I called the script 'BAKER' as shown on the left. This script can export transformation data of vertices and particles into external '.txt' files which will be used to create RenderMan '.rib' files as curves and blobbies. The script can handle multiple selections of any type at once and bakes out frame padded data files within the declared frame range.

In this case, I used the script to bake out data for 4 different particle emitters and 1 mesh object. To bake out data for particle nodes, I turned on the option box for using the particle's RadiusPP attribute as scale values for each blobby. So, attaching 'Spheres' to particles in the Maya scene will be an accurate reference as they will become a same size blobby when rendered.

For the face mesh, position data of each vertex are baked out and random scale values are assigned where it's max-min range can be declared from the interface of the script.


merge_Blobbies
CREATE SINGLE Blobby.RIB FILE SECTION
Browse for Blobby files to Add the list
<<Add to List
Print List Paths -> to view in Shell Window
<<Clear List
Browse Output Directory for single [mergedBlobbies.rib] file
<<Output Folder
<< CREATE SINGLE BLOBBIES.rib >>
CREATE SEQUENCE Blobby.RIB FILES SECTION
Browse for the first Blobby file in Directory [1] -> blobbiesA.0001.rib
<<Read DIR 1
Browse for the first Blobby file in Directory [2] -> blobbiesB.0001.rib
<<Read DIR 2
enter FRAME PADDING (integer only) <<
4
change the FILENAME for output if you like (no padding or extension) <<
mergedBlobbies
Browse Output Directory for [mergedBlobbies.####.rib] Sequence
<<Output DIR
<< CREATE SEQUENCE BLOBBIES.ribs >>
<<Message Log

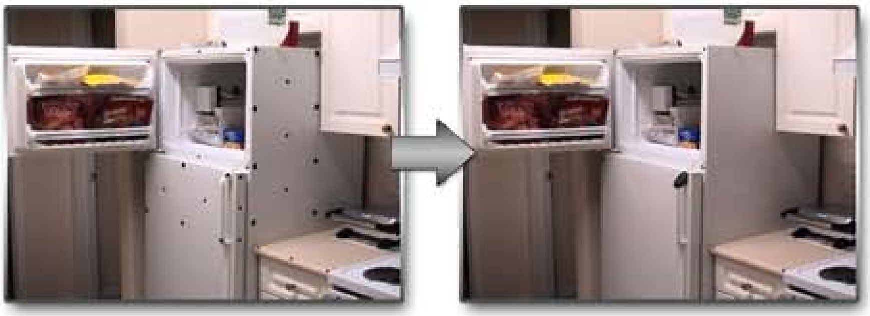
Now, even though I have all the data baked-out as RenderMan blobby '.rib' files, they are baked out as 5 seperate sequences. They have to be merged into one '.rib' file for being able to render as a whole blobby object.

To do this, I wrote a Python script as shown on the right which uses the Tkinter module for the GUI. The script is basically a text handler which can merge multiple blobby '.rib' files into a single '.rib' file. The script has the ability to merge files for each frame within sequences or add a single '.rib' file into a whole sequence.

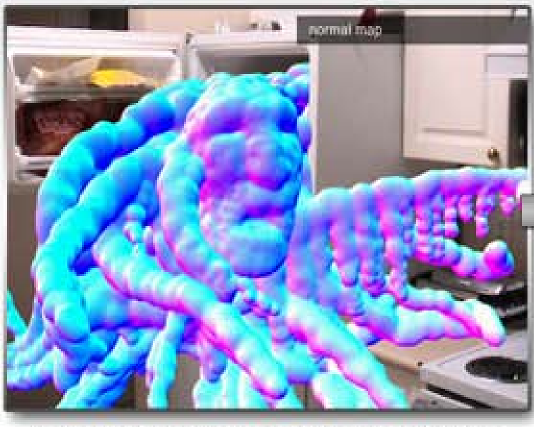
After the final blobby '.rib' files are created they are ready to be sourced and rendered in Maya.

The footage for this project was shot by a non-professional DVcam. So, I decided to use more tracking points for matchmoving as the quality of the footage would be limited. The tracking points are actually ducktapes sticked on the refrigerator as plus signs which are also carefully measured in terms of distance in between them. The track points are deleted for post-production by using the clone tool in Nuke.

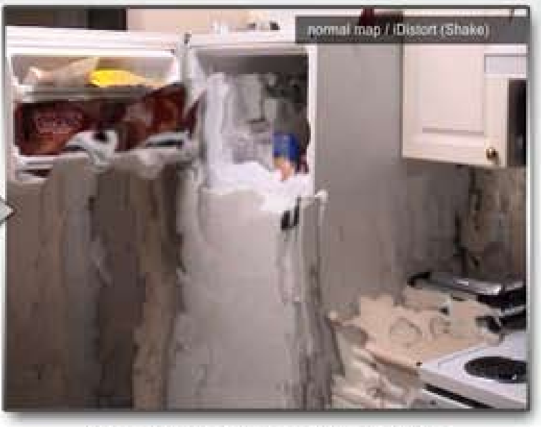
The footage also had a flickering problem as I shot this footage under florescent light. The flickering issue was also fixed by using the DeFlicker plug-in in Nuke. A snapshot from the original footage is shown below on the left and the post-processed version on the right.



An important characteristic of ice material is refraction. However, it is very time-consuming for renderers. So, I decided to use a fake refraction technique to avoid long lasting renders. I wrote a custom RenderMan shader which basically paints the object to red, green and blue based on the normal angle of the geometry. Then, this pass was used in Shake in conjunction with the 'iDistort' node. The result is shown in the next page on the right. Nuke has a similar node called 'iDisplace' but I decided to use Shake's 'iDistort' just for this effect simply because I thought Shake handles this process better than Nuke.



fake refration pass rendered with custom RenderMan shader



final refraction effect with Shake's 'iDistort' node

All the other passes were composited in Nuke which consist of 6 passes.

The beauty pass is rendered by using a custom RenderMan displacement shader I wrote.

The rim pass is also a custom shader written by me which affects the edges of the geometry in terms of specularity.

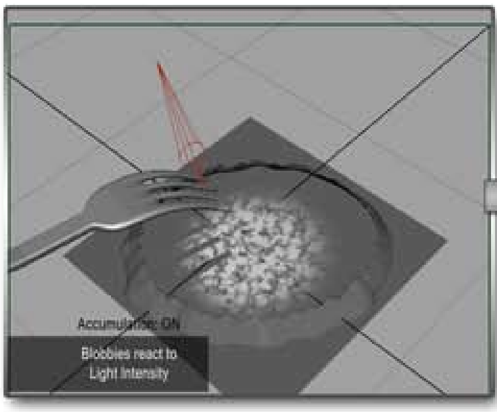
The smoke pass is rendered using Maya Hardware Render Buffer. It is a particle simulation rendered as Multipoints, Multi-Pass rendering and Motion Blur is used.

Other 3 passes are IndirectSpecular, Occlusion and Shadow passes.

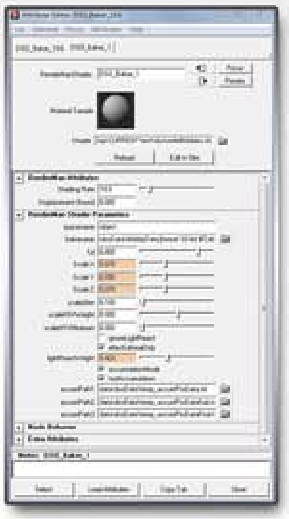
Work Done:
Matchmoving, Shader Writing, Scripting, Compositing

Software Used:
Maya/RenderMan, Nuke/Shake, Boujou

LIGHT-REACT BLOBBIES



In this project, I wrote a custom RenderMan DSO Shader which can place a blobby on each micro polygon of the object. The position of the blobbies depend on the intensity value of the light falling on object's surface which the DSO shader is applied. The shader works in conjunction with a custom DSO plug-in I wrote in C language which lets the shader to bake out blobby data to external '.rib' files. The attributes of the blobbies can be specified from the shader interface. The atributes are listed in next page.



Shading Rate: Acts like a multiplier for the blobby count on the object. The shader looks for micro polygons to place blobbies so a higher shading rate will place less, a lower rate will place more blobbies on the object.

Kd: Determines the diffuse weight of the shader.

Bakename: Specifies the output directory for baking out data.

Scale X,Y,Z: Specifies the scale values for each blobby.

ScaleJitter: Adds randomness to the scale values of blobbies.

ScaleHSVWeight: Declares blobby scale based on the intensity value of light on the surface. Blobbies will become smaller where the light is brighter.

ScaleHSVMin: Any blobby whose scale is smaller than this value will become invisible to the renderer.

ignoreLightReact: If this optionbox is checked, light's effect on blobby placement will be ignored and blobbies will be placed as sticked on the surface.

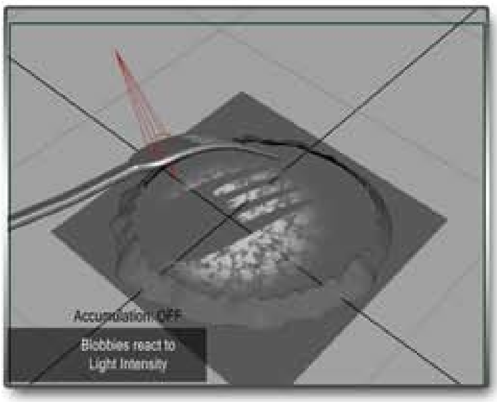
lightReactWeight: Acts as a multiplier to light's intensity falling on surface.

accumulationMode: If this attr is checked on, the shader will keep adding the new data to the previous frame's data. So, new blobbies will be placed in addition to the already existent blobbies for each frame.

fastAccumulation: If this attr is checked off, the shader won't place a blobby to a position where another blobby was placed there before at the previous frames. So, it will be an heavy and time-consuming process not to use fast accumulation as it checks for the position vector of each blobby if it exists at the previous frames or not.

accumPath1,2,3: Specifies the output directories for the files which will be used as references for baking out blobby data if the accumulationMode is on.

The DSO shader picks up the '(V)alue' from object's surface by converting RGB data to HSV which provides great flexibility for the techniques to animate with light. Any object passing in front of the light leaving a shadow on objects surface, any color or transparency maps applied to objects or lights will affect the animation in limitless creative ways. Another characteristic of the shader is that the blobbies will follow the light's position and they will be pulled through to the light source in terms of the normal angle of the light reperesented as (L) in RenderMan.



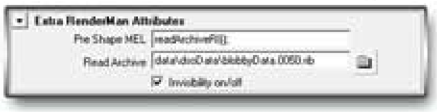
In this case, I assigned a noise texture to the light's intensity map for adding variation. A fork is also passing in between the light and the object, where it's shadow falls on the surface. After animating the light's position, all I got to do was to apply my custom DSO Shader to the object and render. As a result, I had blobbies reacting to the light's intensity and position. However, to use this shader efficiently, certain steps must be taken into account which forced me to write my own custom MEL scripts for building up a pipeline.

Most of the time, the lights that will be used to create the DSO light animation and the lights that will actually light the scene would be different. So, they should be classified for being able to turn on/off when appropriate. Furthermore, the scene should be rendered first by assigning the DSO shader to the object for baking out the blobby data. Then, the scene should be rendered again this time with the desired surface shader assigned to the object where the blobby data will be sourced.

So, the following steps should be followed to render light-react blobbies.

1. Apply the DSO shader to the object.
2. Turn on the DSO lights.
3. Turn off the scene lights.
4. Render to bake out blobby data.
5. Source the blobby data to the object.
6. Apply desired shader to the blobby sourced object.
7. Turn on the scene lights.
8. Turn off the DSO lights.
9. Render again for the final look of the blobbies.

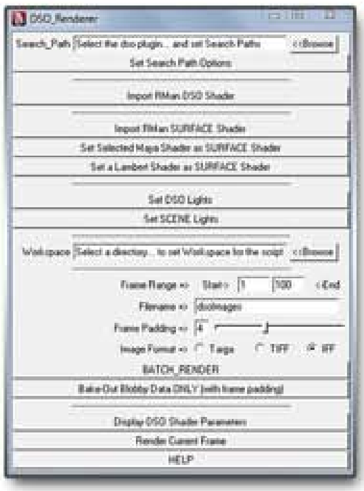
To source the blobby '.rib' files, I wrote custom RenderMan Studio(RMS) scripts consisting of two MEL scripts and a '.rman' script for being able to render pre-baked '.rib' files when assigned to an object. The UI will appear in the Attr Editor of the object.


Extra RenderMan Attributes
Pre Shape MEL: readArchiveFill();
Read Archive: data\dsoData\blobbyData.0050.rib
[✓ Invisibility on/off]

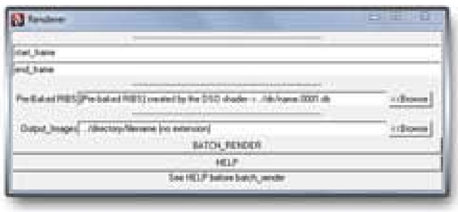
Obviously, doing all these steps manually would be an hassle. So, I wrote the 'dsoRenderer' MEL script shown on the right which does all these steps at once just by setting a few parameters and pressing the 'batch_render' button.

You can set the search path for the renderer to see the DSO plug-in, import the DSO shader, declare the surface shader, classify dso/scene lights, render a test frame and batchrender all from the interface of the script. Also parameters like frame range, workspace directory and the properties of the final image can be declared from the interface.

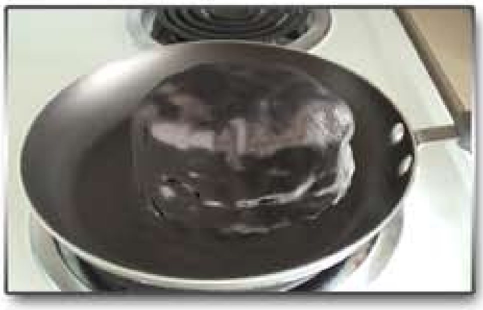
Another feature of the script is the ability to bake out blobby data files with frame padding. In this option, just the blobby data files wil be baked out and no images will be rendered.

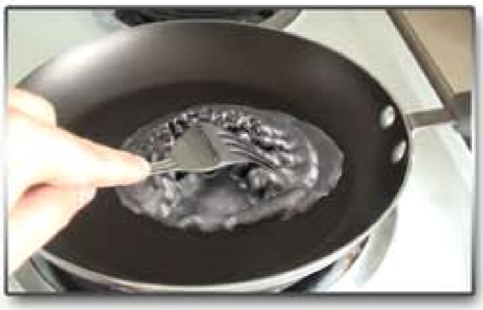

DSO_Renderer
Search_Path: Select the dso plugin... and set Search Paths
Workspace: Select a directory... to set Workspace for the script
Frame Range => Start=> 1 100 <End
Filename => dsoImages
Frame Padding => 4
Image Format => Targa TIFF [✓ IFF]

<Browse
Set Search Path Options
Import RMan DSO Shader
Import RMan SURFACE Shader
Set Selected Maya Shader as SURFACE Shader
Set a Lambert Shader as SURFACE Shader
Set DSO Lights
Set SCENE Lights
<Browse
BATCH_RENDER
Bake-Out Blobby Data ONLY (with frame padding)
Display DSO Shader Parameters
Render Current Frame
HELP


Renderer
start_frame
end_frame
Pre-Baked RIBS: [Pre-baked RIBS] created by the DSO shader -> ../rib/name.0001.rib
Output_Images: ../directory/filename (no extension)

To be able to render images by sourcing these frame padded files, I wrote a MEL script called 'renderer' on the bottom right. This script will work in conjunction with the RMS scripts explained above and increment the frame pad of the sourced '.rib' file under the 'ReadArchive' attribute one by one for each frame.





The matchmoving for this footage was made using Boujou. The original footage has trackable plus shaped tapes sticked on the pan which are cloned later in Nuke for the final image.

The blobbies shown on the right are rendered by turning on the 'ignoreLightReact' option from the DSO shader at these frames while baking out blobby data. In the Maya scene, there is just a cylinderical shape moving up on Y axis and because that the lights didn't affect the blobbies as a result of the 'ignoreLightReact' option the blobbies sticked on the surface within these frames.

The final image was rendered using RenderMan by sourcing the blobby '.rib' files with the aid of my scripts. A custom displacement RenderMan shader and a reflective Blinn shader was used on the blobbies. There are 4 main passes used for compositing in Nuke as specular, diffuse, reflection and occlusion. The reflection of the blobbies on the pan is also rendered as a seperate pass.

Work Done: Matchmoving, Shader Writing, Scripting, Compositing, C Programming

Software Used: Maya/RenderMan, Nuke, Boujou

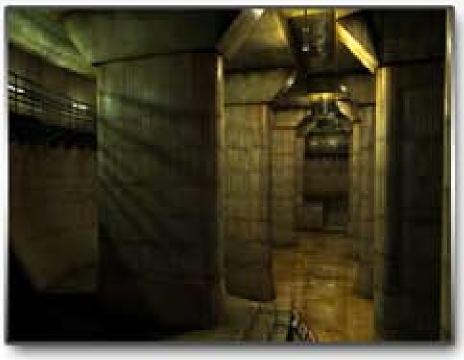
SEWER

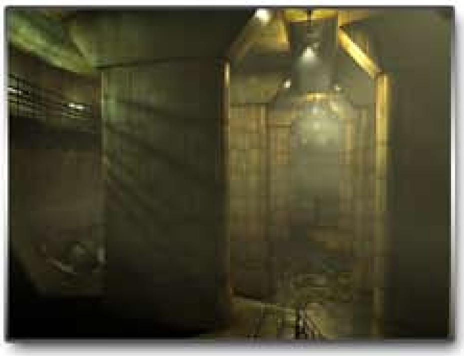
I thought of this structure as an unusual, huge underground sewer. The scene consists of 12 pointlights and 4 spotlights. Pointlights are used for the lighting of the lamps attached to the side wall and quadratic lighting is used in spotlights to enhance the feeling of the huge size. There is also slight green/yellow variations in the lights' color to give this moody underground feeling. The renderer used for the beauty pass was Mental Ray.

The fluid simulation is created with RealFlow. I thought of this fluid simulation as a rush of waste oils into to the sewer. So, I kept the viscosity a little lower somewhere between water and oil and used brownish colors to give the feeling of waste.

Maya particles are added to the fluid simulation as seperate passes where the fluid hits the column walls. And the trace left by the fluid on the side wall is a particle simulation rendered as mental ray blobbies.

I also added another Maya fluid simulation as the fog and applied it on the top of the scene to emphasize the height of the structure. It is a fairly thick fog driven by fluid turbulence and vortex fields. Also, Maya environment fog is attached to the ceiling lamps to enhance the feeling of a foggy environment.



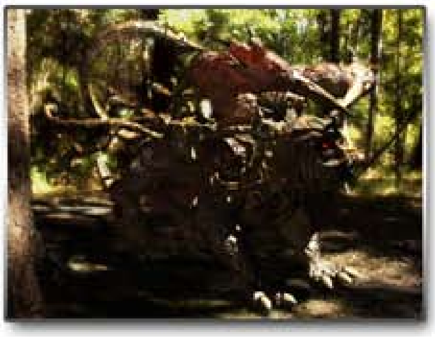


Work Done: Modeling, Lighting, Shading, Texturing, Compositing, Rendering

Software used: Maya/Mental Ray, Nuke, RealFlow

VULTIGRA

My goal in this project was to integrate a CG character into a real photograph. I had created this tiger-like creature and I wanted to give it a feeling of ancient times sculpture mixed with modern technology look. The skin has subsurface scattering with a backScatter, a bump and a displacement map applied. Procedural shaders are used mostly on skin and eyes. It is all rendered in Mental Ray by using the rendering in layers method later to be composited in Shake. It consists of 3 main passes as beauty, shadow and occlusion.

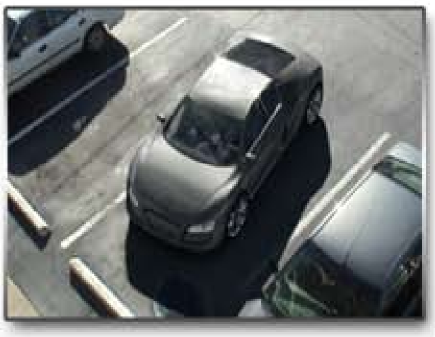


Work Done: Shooting, Lighting, Texturing, UV Layout, Shading,Rendering, Modeling, Compositing

Software Used: Maya/ Mental Ray, Shake

AUDI LE-MANS

My goal in this shot was to get the fastest but fairly low quality renders and composite them using 'rendering in layers' method to make it look high quality realistic. The longest pass was the occlusion pass which was rendered in 8 seconds per frame. All other at about 30 passes were under 3 seconds per frame. The footage was shot with a 2.0 megapixel Sony HandyCam without any markers put on the scene and tracked using RealViz.



Work Done: Shooting, Matchmoving, Lighting, Modeling Shading,Rendering, Animation, Compositing

Software Used: Maya/ Maya Software, RealViz, Shake

ROLEX WATCH

This Rolex watch was rendered with MentalRay. No other compositing or image manipulation software is used afterwards. The result has derived from a combination of different lighting techniques as HDR, Final Gathering, Global Illumination and Caustics that pretty much covers all lighting in Maya/Mental Ray. I had also improved my photon mapping techniques in this project but choosed not to use it for the final look of the render.

Work Done:
Modeling, Shading, Texturing, Lighting, Rendering

Software Used:
Maya/Mental Ray

CGI TWIN

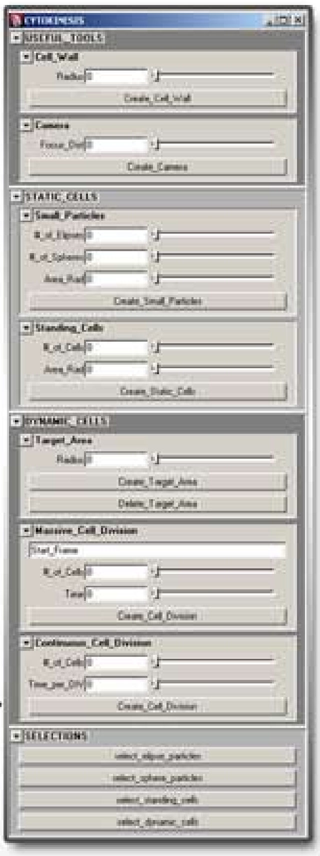
In this shot I created a CG twin of the lock from the reference photo I shot and integrate the CG version next to it by mimicking the environmental characteristics of the photo. 'Rendering in Layers' method is used to render seperate passes which are Specular, Diffuse, Reflection and Shadow all rendered using Mental Ray. All 4 passes were composited using Shake.

Work Done:
Modeling, Lighting, Shading, Texturing, Rendering, Compositing

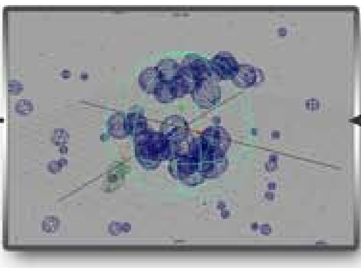
Software Used:
Maya/Mental Ray, Shake

CYTOKINESIS

My goal in this MEL script was to create a tool for the artist to simulate massive cell divisions as fast and easy as possible. The script is based on creating a single cell division at the origin of the scene and moving it to a random position within the declared target area which can be controlled from the interface of the script. One of the main feature of the script is to create 'massive cell division' where all the cells will start dividing at the specified frame and last for a specified amount of time. Another main feature is to create 'continuous cell division' where the cells will divide one after another in between the specified amount of time. Other features include creating big amounts of 'static cells' and 'small particles' within a few clicks, 'easy selection of each type' and creating a pre-setup camera with the best rendering options. The shaders used on the cells are custom RenderMan shaders that I wrote and the final render is manipulated in After Effects just for color correction and glow effects.



custom RenderMan shaders applied



created in 10 - 15 mouse clicks

Work Done:
Scripting, Shading, Rendering, Compositing

Software used:
Maya, RenderMan, Adobe After Effects