A physically-based 35mm stills camera simulation

Tom Griffiths
National Centre for Computer Animation
Bournemouth, UK

Abstract

Although there has been extensive rendering research into the simulation of real camera behaviour such as depth of field and motion blur, due to the increased render times these behaviours inherently bring, they are all simulated approximations.

In this report I describe a physically–based 35mm camera simulation, implemented in the Maya software package. More precisely, a physically-based camera accurately computes the irradiance on the film given the incoming radiance from the scene **[1]**.

In my simulation the camera is made up of a lens system, camera body, film plane and shutter mechanism. The lens system consists of three lens elements and an aperture stop. The shutter mechanism is made up of two shutter curtains and the film gate. The irradiance on the film is calculated by tracing rays through the lens system (from the film plane) and retrieving the radiances from the scene. As I am trying to accurately simulate a real camera, optimization is not part of the project and render efficiency is not a concern.

1 Introduction

One of the short-comings of current camera models is due to the rendering optimization of tracing rays from the camera, rather than from their true origin, the light source. Although this severely cuts down on render times, it means the implementation of the exposure on the film backplane (the output of the rendering system) is done counter-intuitively by adjusting light levels, whereas in reality it would be done by adjusting camera settings to correctly expose the film.

Although it is currently possible to adjust aperture sizes (or f-stops) for the calculation of depth-of-field (implemented differently in different rendering systems), they do not take into account the geometry of these apertures and therefore do not correctly affect the exposure. Another important element of the exposure system, the exposure time, is not currently implemented.

There are two situations in which a correct camera simulation would be very useful. The first is in the case of using the rendered image for the purposes of compositing it with real camera footage/photographs. The exact specifications of the real camera could be modelled within the 3D package, producing a render specific to that camera type and reduce the amount of work required in post-production (balancing of exposure for example). The second situation, and the primary reason for my research, is to teach users how to use a camera correctly. Due to the controllable nature of the scene within the 3D package, and the lack of

necessity for the inner mechanical workings of the camera, the learning process should be more intuitive.

In this report I will describe a physically-based, 35mm stills camera simulation, implemented in the Maya software package. Although the system uses a specific lens type (see fig 1), it is possible to use other lens types, so long as the technical specifications are known in order to model the lens. Using different lenses would also enable different sized film planes (the size of the film negative/positive) to be used so technically, this method can be used to create any known camera model (so long as the relevant technical data is available).

My aim was to create a camera model that worked in the same way as a real camera in that it allows the user to set aperture and exposure time. Focusing, focal length and film iso rating will be discussed but were not directly implemented in this example.

I will begin by explaining how the lens system was created and how it works within the model. Next I will explain how the film plane was implemented and how it works with the lens system. From this base model, the aperture will be presented and, in conjunction with the setting of focal point and focal length, its affect on the image's depth-of-field and exposure will be explained. Next, exposure time (shutter speed) and its application will be introduced and finally, some results from my simulation will be shown and discussed.

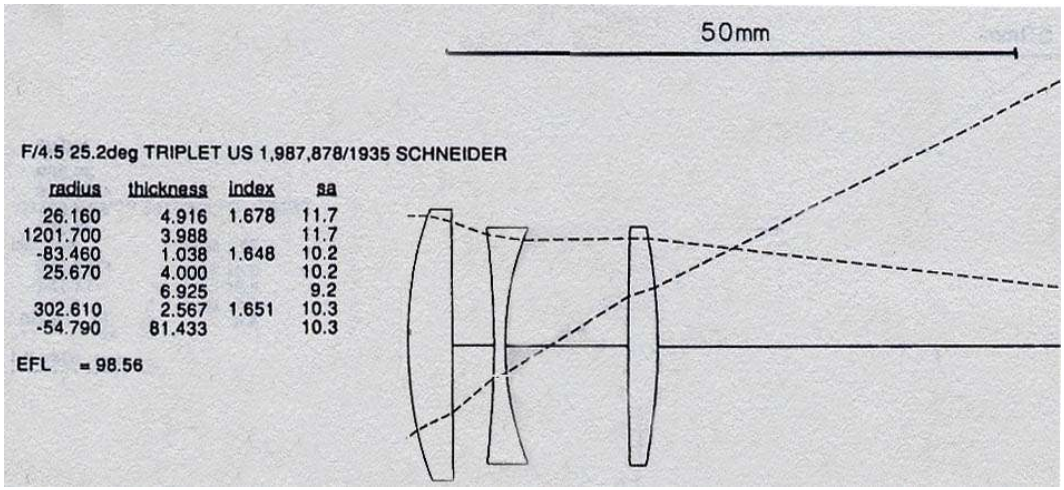


Fig 1: Diagram showing the lens used in the simulation. Each row in the table on the left describes a surface on a lens element. The surfaces are listed in order from the left most surface in the diagram. All measurements are in millimeters except for index of refraction. The first column gives the signed radius of a spherical element, if none is given, the surface is planar. Positive values mean the surface is convex and negative values mean in is concave. The second column gives the distance from one surface to the next, along the central axis. Index refers to the index of refraction for the lens element (the current surface and the one below). The last column, sa, is the diameter of the surface. The column that has no radius listed is an air space (where the aperture stoop would go). The sa value for this row refers to the diameter the aperture stop would have when fully open.

2 Lens System

Camera lens systems are made up of a number of glass lens elements (or plastic, depending on the quality of the lens), that are centrally aligned. Each lens element is spherical and is either concave or convex in profile on each surface. Within these elements, the aperture stop is housed. The aperture stop is a mechanical element that is used to limit the amount of light entering the camera **[1]**. The aperture is discussed further in section 4.

For the purposes of this project, I used a lens of the Cooke triplet design type with data obtained from the book by Warren J.Smith (see Fig 1) **[2]**. This particular design is the type used for inexpensive 35mm lenses, and is therefore ideal for my uses.

To create the lenses, I started with NURBS spheres, with the correct radius obtained from the lens data table in fig 1, then used Booleans between two of these spheres, to create a solid with the correct surface profiles. Finally, Booleans were used between this solid and a cylinder to get a lens with the correct diameter (see fig 2).

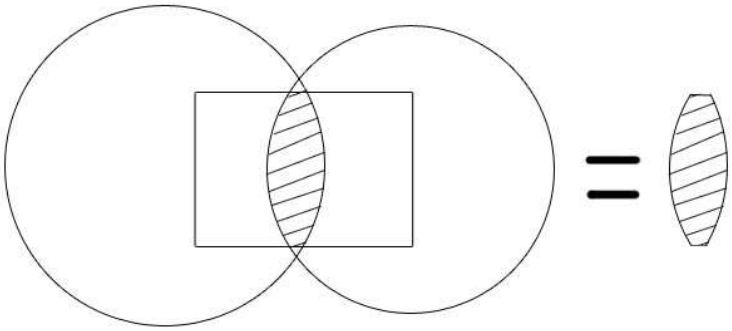


Fig 2: Example of how to use booleans to make a lens (profile view). By using an interception boolean on two spheres and a cylinder (the rectangle), the result is the lens (shaded in stripes in the diagram).

The purpose of the lenses in a camera is to refract the light that hits them, into the camera body towards the film, thus resizing the incoming light image to the size of the film negative/positive. However, real lenses don't just refract the light, they also reflect a portion. If light gets internally reflected within the lens elements, when it reaches the film, and if it is intense enough, it can cause lens flares. I decided not to implement this into my model, as it doesn't affect the way in which the camera works, with a view to implement it later, time permitting, as it would supply more realistic results.

Ideally, a camera would only need one lens to produce the image on the film plane (with the exception of telephoto lenses that can change the overall magnification of the subject in image space). The reason cameras use more than one lens is because the glass (or plastic) used refracts light of different wavelengths differently, causing aberrations. The job of the each subsequent lens is to correct the aberrations caused by the previous one. These aberrations are particularly evident in wide-angle lenses in the form of chromatic aberration (when objects appear to have a ring of different colours, mainly red and blue, around their edges). See fig 3.

Fig 3: Example of chromatic aberration. Blue edges are clear on the left with red edges on the right. (Source: http://www.dpreview.com/reviews/AdobePhotoshopCS2/images/CA_01_ORIG.jpg accessed 2nd March 2006)

As I am using the Renderman rendering system, I had to write a shader for each of the lenses (they are basically a modified version of the glass shader example in the Renderman documents. **[3]** See fig 4).

```
Surface
front_lens (
        float ior = 1.678;)                     /* sets the index of refraction */
{
        normal Nn = normalize(N);               /*Normalises the normal vector*/
        vector In = normalize(I);               /*Normalises the incident ray vector*/
        normal Nf = faceforward(Nn, In, Nn);    /*Makes the normal of the surface face the camera*/
        vector refrDir;
        float eta = (In.Nn < 0) ? 1/ior : ior;  /* Calculates the relative index of refraction */
        Ci = 0;                                 /*Sets out colour to black*/
        refrDir = refract(In, Nf, eta);         /*Redirects the incoming ray*/
```

Fig 4: The shader applied to the front lens element. All other lens use the same code except the index of refraction (at the top) is different.

Each lens utilises the same basic code, but uses a different index of refraction (as specified in the lens data table, fig 1). In a real camera, the lens redirects incoming light rays according to its index of refraction (the larger the index of refraction, the larger the change in the direction of the ray), whereas in the computer model, due to the fact that rays are cast *out* from the camera, rather than from the light source, the job of the lens shader is to refract the camera rays out into the scene.

3 Film Plane

When the light rays that have been refracted through the lenses reach the film plane, it reacts with the photosensitive emulsion of the film, creating a latent image (this 'image' is invisible

until the film is developed). The film emulsion is made up of silver halide grains. When a photon from the light hits one of these grains, it transfers energy to an electron from the halide ion and liberates it. This electron then attracts a silver ion and makes a stable silver atom. These silver atoms are what make up the latent image.

In the computer model, this latent image doesn't really exist, only in the time between the calculation, and the display of the rendered image. The principal of latent image still applies though. For real film, the more light that falls on the emulsion, more stable silver atoms are created, making a denser image (more saturated in the case of colour). I needed to simulate this process in the computer and the best way of doing this is with ray tracing. As light isn't just cast in a single direction, but in all directions, it means that light from an arbitrary point in front of the camera will have multiple light rays entering the lenses from multiple angles. This in turn means that for each point on the film plane, multiple rays will fall on them, thus having an additive effect. In actual film, this would be increased energy, but as Maya doesn't currently model the physical energy of light, simply the colour and its intensity, the additive effect on the simulated film will be that of colour and its saturation (the more rays of a certain colour that fall on a point, the more saturated of that colour the point will become).

To simulate this effect in my camera model, I wrote a Renderman shader that casts rays out from each point on the film surface, in multiple directions (see fig 5).

```
Surface
film_plane(float rays = 64;      /*Shader accuracy (Number of rays shot from each point)*/
        float angle = PI/2;)   /*Rays are cast within a hemispherical cone (within 90 degrees from the normal)*/
{
        normal Nn = normalize(N);
        Ci = 0;                /*Initial colour is black*/
        color retrieved = 0;   /*Colour value retrieved from ray*/
        gather("illuminance", P, Nn, angle, rays, "volume:Ci", retrieved){
                               /*Add colour from ray to surface colour total so far*/
                Ci += retrieved;
} else {
                               /*If the ray doesn't hit a surface, add black to the total colour*/
        Ci += 0;
```

Fig 5: The shader used to simulate the film emulsion.

The gather function used is a built-in function of Renderman that casts rays out from a given point on the film plane, in directions subtended by the sample cone. When these rays hit an opaque surface, they return the colour value of that surface at the strike point (after the effect of lighting has been calculated). The value of the colour returned by the ray is added to a running total of all the other rays cast out from that point. This total is then averaged by dividing it by the total of rays cast. This means that increasing the amount of rays used will only increase quality, and not change the brightness of the resultant image (hence making the exposure process easier to judge).

The camera model is now at a point where it can take pictures but the results are no different from the current camera model employed in Maya (although they take a significantly longer time to render). In order to obtain real camera results, the aperture stop has to be introduced.

4 Aperture

The aperture stop (or sometimes referred to as the iris) is made up of metal leaves that are layered in a fashion so that when they are rotated inwards, the hole they encircle (the aperture or entrance pupil) becomes smaller, thus reducing the amount of light that passes through it.

Each full stop lets half the amount of light through its entrance pupil than the one previous to it (the f-number is increasing). This is because the area of the entrance pupil is

$d1 = 1^{st}$ Diameter (Smaller f/stop)
$d2 = 2^{nd}$ diameter (Larger f/stop)

$$\pi (d2/2)^2 = \tfrac{1}{2} \pi (d1/2)^2$$
$$(d2/2)^2 = \tfrac{1}{2} (d1/2)^2$$
$$2(d2/2)^2 = (d1/2)^2$$
$$2 = (d1/2)^2 / (d2/2)^2$$
$$2 = (d1^2 / 4)/(d2^2 / 4)$$
$$2 = d1^2 / d2^2$$

half that of the previous and therefore means the diameter of the pupil is decreased by a factor of $\sqrt{2}$ (see fig 6).

Fig 6: Equation to show that the diameter of an aperture is scaled by a factor $\sqrt{2}$ when changing between subsequent f/stops.

The f/stop is the ratio of the focal length of the lens to the diameter of the entrance pupil.

f = F/D

f/stop = focal length/diameter of entrance pupil

This works out to the f/stops corresponding to the sequence of the powers of $\sqrt{2}$. i.e. f/1.4, f/2, f/2.8, f/4, f/5.6, f/8, f/11, f/16, f/22 (the numbers are rounded off to make them easier to write down, but this is how they appear on modern cameras). See fig 7.

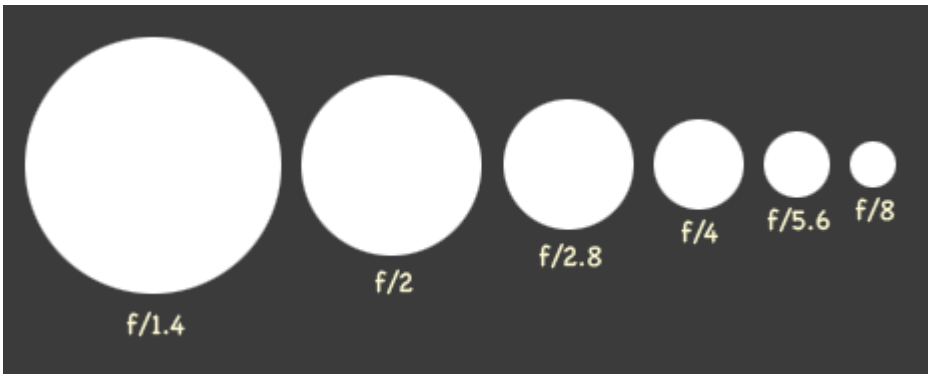


Fig 7: Examples of f/stop (aperture) values with comparative areas.
(Source: http://en.wikipedia.org/wiki/Image:Aperture_diagram.png accessed 2nd March 2006)

The size of the aperture also has an important effect on the depth of field of the resultant image (the portion of the image on the film plane that is within acceptable levels of focus). The bigger the aperture (smaller the f/stop), the shallower the depth of field. The reason for this is due to the fact that larger apertures let in more light, from a greater number

of angles. When a point in space is not focused on by the lens, it means its image is not focused onto the film plane (the light from it does not converge onto a single point). Effectively this means the light from it falls over an area of the film plane, rather than a point. As the aperture gets bigger, it lets light from a greater angle enter the camera, thus falling onto a larger area on the film plane (see fig 8). As the light from a point is falling over an area, it is dissipating its energy and creates a blurred image. Because of this energy distribution, the blurred image is darker than it would be if it were in focus.

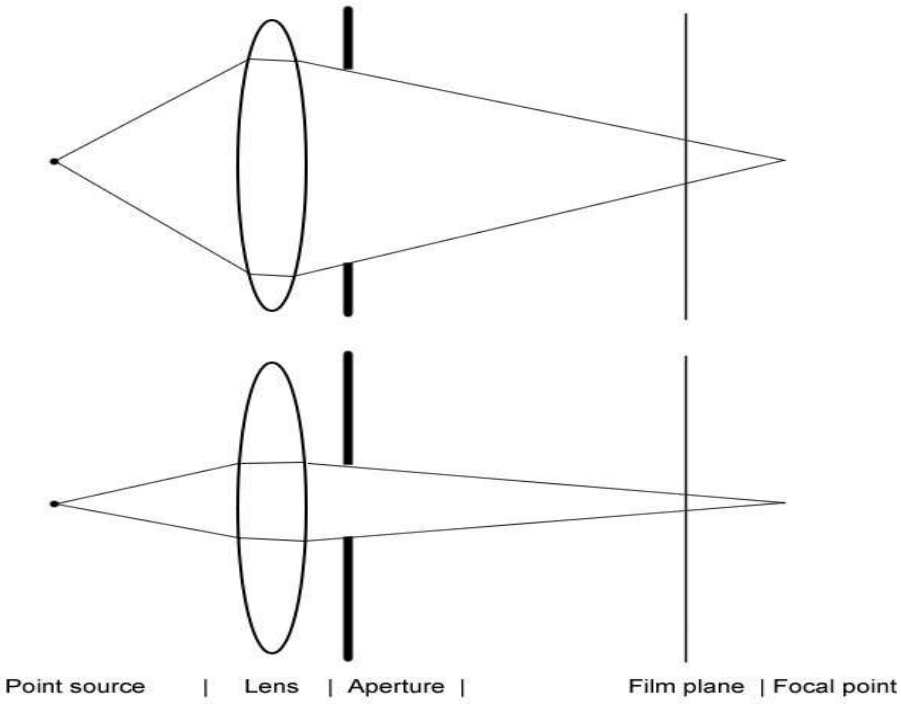


Fig 8: The top diagram shows the path of light rays from a point source through a wide aperture. The bottom diagram shows the path of light rays through a narrow aperture. It is clear to see that the area the light covers on the film plane with a narrow aperture is smaller than the one with a wide aperture.

The last thing that the aperture affects is the shape of out-of-focus highlights, also known as the Bokeh effect **[4]**. The area that light from a point covers on the film plane is called the circle of confusion. When the circle of confusion becomes larger than the resolution of the eye, the point is deemed to be out of focus. If the light from the given point is significant enough in intensity, and it is sufficiently out-of-focus, its corresponding circle of confusion becomes noticeably visible. As the job of the aperture atop is to limit the light entering the camera, the cone of light that does manage to enter is in the shape of the aperture. For example, if the aperture stop uses 6 leaves, it will mean the light cone entering the camera will be hexagonal and out-of-focus highlights will also be hexagonal (although it sounds strange, this will mean that the circle of confusion in this case would be hexagonal. It is only called the circle of confusion because without an aperture stop it would be a circle, and apertures with a larger number of leaves will produce more circular results). Although this effect is only visible when a highlight is bright enough, it does in fact happen to every light cone that enters the camera, it's just that the circles of confusion overlap to produce a seemingly uniform blur (see fig 9).

Fig 9: The hexagonal highlights on the wine glass in the background are clearly visible in this photo. The blur in the rest of the out of focus part of the photo is uniform. Photo taken by Rick Denny
(Source: http://www.rickdenney.com/images/CRW_2796.jpg accessed 3rd March 2006)

Depth of field is not just affected by the aperture. The focal length of the lens used also has an affect. As a rule, lenses of a shorter focal length (wide angle lenses) have a larger depth of field than long focal length lenses (telephoto lenses). The reason for this is due to the fact that short focal length lenses have a smaller magnification effect than long focal length lenses. This means that, relatively speaking, the image of the incoming light from a point is smaller when using a short focal length (providing the camera does not change position). This in turn means that the light cone emanating from the last element in the lens has a smaller angle, thus producing a smaller circle of confusion and a sharper image.

The distance the lens is focused at also affects the depth of field. When the focus distance is short, the depth of field becomes shallow. When it is focused far away the depth of field becomes deeper. Again, this has a similar reason to using a wider aperture. If a point is close to the camera, the angle of the incoming light cone will be quite wide, thus the projected light cone will be wide and produce a large circle of confusion. Changes in distance when close to the camera will mean a relatively large change in the cone angle. As the point gets further away, the change in angle becomes smaller (assuming the same aperture is being used). This means the focus points (inside the camera) from objects far away, will be closer together than focus points from objects close to the camera.

In order to make the aperture in my system, I first modelled a single aperture leaf and duplicated it 5 times, putting the leaves in a circular arrangement, thus creating a hexagonal aperture. Each leaf has is pivoted in a way so that when they are all rotated towards each other, the aperture they subtend becomes smaller, but stays the same shape (see fig 10). Aperture stops in real cameras can have any number of leaves and the more there are, the more circular the aperture becomes. Unlike mine, the leaves tend to have a curved edge so that even with a small number of leaves, the aperture will still resemble a circular shape.

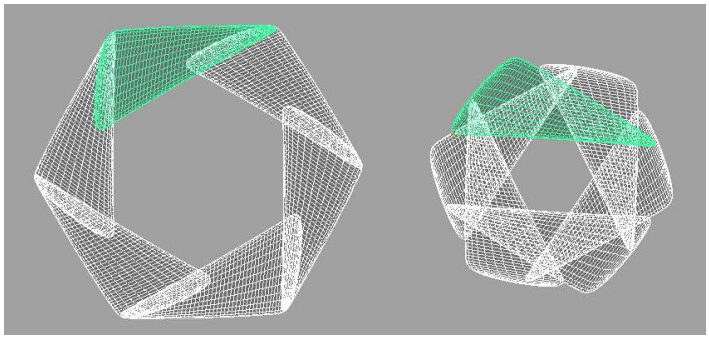
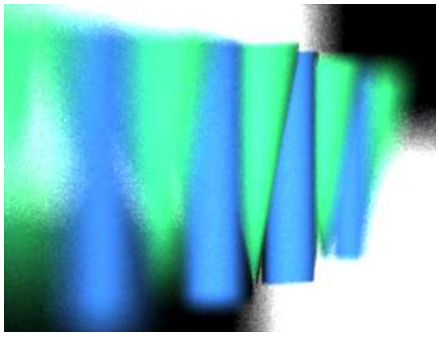


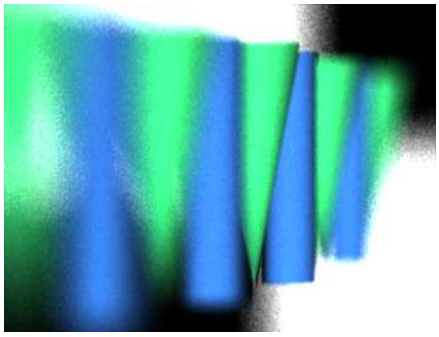
Fig 10: The model of the aperture leaves. The image on the left shows the aperture fully open. The image on the right shows what the aperture looks like when it becomes smaller.

As my system is a model of an ideal camera, I decided to make the insides of the camera completely black so that the light that enters the camera doesn't get reflected and bounce around. If there is reflection within the camera, it can cause fogging of the image or produce artifacts similar to lens flares. To make sure there was no reflection in my model, I used a constant black Renderman shader. This means that if a ray from the film shader hits the black surface, it won't add any colour value, meaning the image stays more faithful to the scene it is representing.

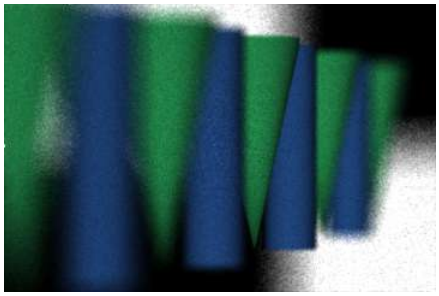
Initial results



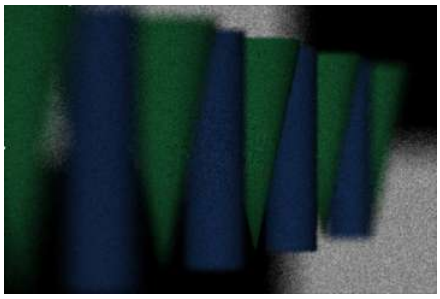
f4.5



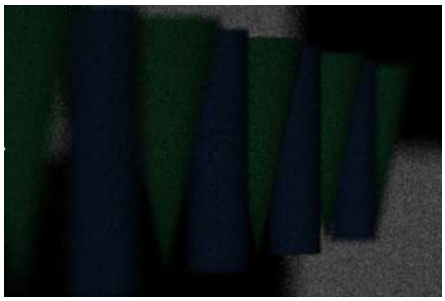
f5.6



f8



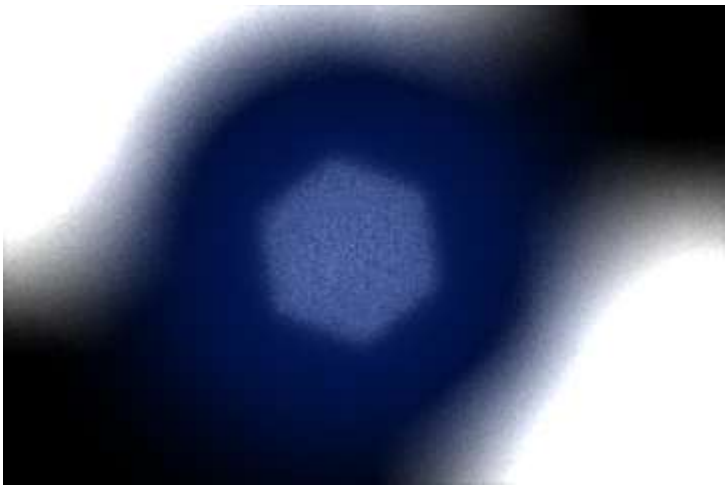
f11



f16

f22

The pictures above are the renders the camera currently produces (with their corresponding f/stop noted underneath). Each stop down (increasing the f-number) produces an image that is half as bright as the stop above, just like in a real camera (with the exception of f5.6 as it is not a full stop down from 4.5). Also, the image of the scene is flipped both horizontally and vertically, with respect to how it would be viewed by the eye. The final thing to note about these renders is that the depth of field also increases as the aperture gets smaller (f-number gets larger). At this stage the only way to change the exposure is to increase or decrease the light intensity. To complete the simulation, the aspect of time has to be implemented.



Above is a render of a blue sphere against a checked background. Both the background and sphere are a distance away from the focal point and out of the depth of field. The highlight on the sphere correct simulates the Bokeh effect described above by taking on the shape of the hexagonal aperture.

5 Shutter Speed

The last exposure effecting element of the camera that needs to be implemented is the shutter speed, or more precisely, the exposure time. As the film is exposed to the light for longer, the amount of energy transferred increases (more photons hit the film emulsion). In physical terms this means more electrons are liberated from the halide crystals, producing a denser image (meaning the image is brighter). In terms of my simulation, this means that a greater exposure time will allow more rays to to strike a surface and return it's colour. However, this must be implemented over a time period due to the fact that more rays cast at a given time will

only produce a better quality image rather than a brighter one (see section 3). The logical way to do this is to render a number of frames with the desired aperture and add them together afterwards. This affectively means there is a constant flow of rays being fired out from the film shader (just as light rays area a constant flow of photons).

Similar to the f/stop, each step down in the exposure scale will let in half as much light than the previous step. On the camera this will mean that the number value will be increasing. The reason for this is because the numbers actually refer to a fraction of a second, and are just written in this way to make it easier to write down. e.g. 125 on the exposure scale actually means 1/125 in seconds. Unlike the f/stop scale, the exposure scale is linear and each step down is half the time of the previous (again though, they are rounded for ease of writing). Depending on the camera, the scale is as such 1000, 500, 250, 125, 60, 30, 15, 8, 4 ,2, 1 (some cameras can do exposures faster than 1/1000 of a second). When the scale gets down to 1 (one second), the convention changes from being a fraction to just being the time in seconds, usually indicated with numbers in a different colour to help differentiate them (see fig 11).

Fig 11: An example of a shutter speed dial. The values in yellow refer to full seconds whereas the numbers in white refer to fractions of a second. The number in red indicates this is the cameras flash sync value.
(Source: http://jrp.nikonians-images.com/f4_images/shutter_speed_dial.jpg *accessed 5th March 2006)*

Despite the name, the numbers on the dial don't refer to the time it takes for the shutter to open. The value is actually the amount of time the film is exposed to the incoming light. The actual time it takes for the shutter to open (and similarly close) differs with the make and model of camera but it is known under the common name of the “Flash Sync”. The flash sync refers to the slowest shutter speed that will allow the film gate to be fully open at a point so's to let all the light produced from a flash hit the entirety of the film emulsion. If part of the shutter was covering the film emulsion, only the uncovered area would receive the light and therefore create an uneven exposure.

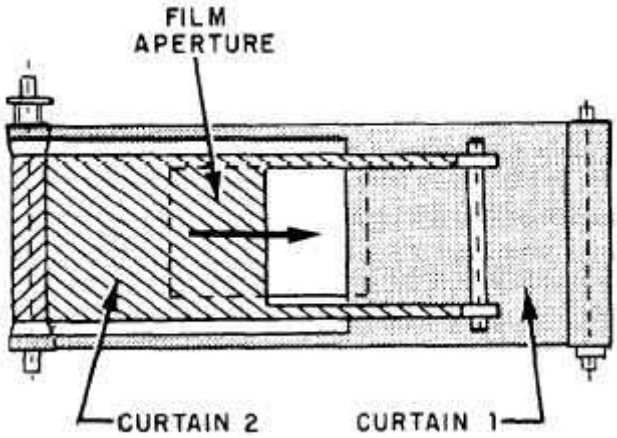


Fig 12: The focal plane shutter mechanism. It consists of two curtains, the front (curtain 1) and the back (curtain 2). The film gate (or film aperture in the diagram) is the opening which allows the light to pass through to the film. (Source: http://www.tpub.com/content/photography/14209/img/14209_89_1.jpg accessed 4th March 2006)

The shutter mechanism is made up of two curtains (front and back) and the film gate (see fig 12). The film gate is a rectangular hole that allows the light through to the film plane. In the case of 35mm photography, this opening measures 36mm x 24mm. The curtains are solid, either made of cloth or metal, and move across the film gate. It is the curtains that stop light from hitting the film plane.

When a photo is taken, the front curtain moves across the film gate thus exposing the film emulsion to the light. After the time specified by the shutter speed dial is reached from the point when the front curtain started to move, the back curtain starts to move to cover the film gate. The total time it takes for the front and back curtains to move from one side of the film gate to the other is equal to the flash sync plus the exposure time (set on the shutter speed dial). For example, if the flash sync was 1/60 seconds and the exposure time was 1/60 seconds, the total time for the photo to be taken would be 1/30 seconds (2/60 seconds).

As both the f/stop and shutter speed allow the camera operator to either half or double the amount of light entering the camera, they can be set in numerous combinations to produce the same exposure (with respect to brightness). This flexibility allows the operator to make creative decisions as to how much of the subject they want in focus. Using a small f/stop (large aperture) will limit the depth of field to produce pleasing out of focus effects, but it will also let a large amount of light enter the camera. To stop too much light entering and causing over-exposure, a fast shutter speed would be used. Similarly, if the operator wanted deep depth of field (a larger area of sharp focus), a small aperture would be used in combination with a long exposure time to avoid under-exposure. For example, a photo taken at 1/125s with and aperture of f4 (the f/stop) would have an equal exposure to a photo taken at 1/30s with an aperture of f8. The difference being that the photo taken with the aperture of f4 would have less depth of field. An example of this is shown in fig 13.

Fig 13: Two images taken with different aperture and shutter speed combinations. The image on the left is using larger aperture than the right image (it is also using a shorter shutter speed to balance the exposure). It is evident that the writing on the glass in the foreground is more blurred in the left image, as is the blue light at the top of the frame in the background. Photos : Authors own.

With aperture the operator has creative control over depth of field and similarly with shutter speed the operator has creative control over motion blur. Motion blur is caused when an object moves with respect to the camera over the period that the film is exposed (an object in view of the camera). Because the object is moving, the light it radiates is changing position and therefore its corresponding image on the film plane also moves. However, as the light from the object falls onto the film for a short period, it doesn't transfer much energy and only produces a faint image (If it is a bright object it will make a more solid image). Also, light from objects that are briefly obscured will have a larger effect on the film as the light from their position is constant and will seemingly overwrite the image of the moving object, making it appear transparent.

Fig 14: Left: Example of a photo with motion blur caused by the movement of the subject. Photo taken by Jeremy Ginsberg. (Source: http://dogmouth.net/photos/misc/mid/2001-05-foothills-maggie-motion-blur.jpg accessed 7th March 2006)
Right: Example of a photo with motion blur caused by movement of the camera.
(Source: http://en.wikipedia.org/wiki/Image:Truck_with_motion_blur.jpg accessed 7th March 2006)

There are two types of motion blur (see fig 14). The first is blur caused by an object moving in front of the camera, leaving a faint streaky image behind it. The second is blur

caused by moving the camera while exposing the photo. This has the effect of making all objects leaving a faint image behind them. However, if the camera is moving to follow a moving object, the moving object will appear stationary and to a degree, unblurred. This does however depend on whether the camera correctly matches the speed of the object. If the camera movement is too slow, the object will leave a trail behind it. If the camera is too fast, the object will have a trail in front of it (with relation to its direction of movement).

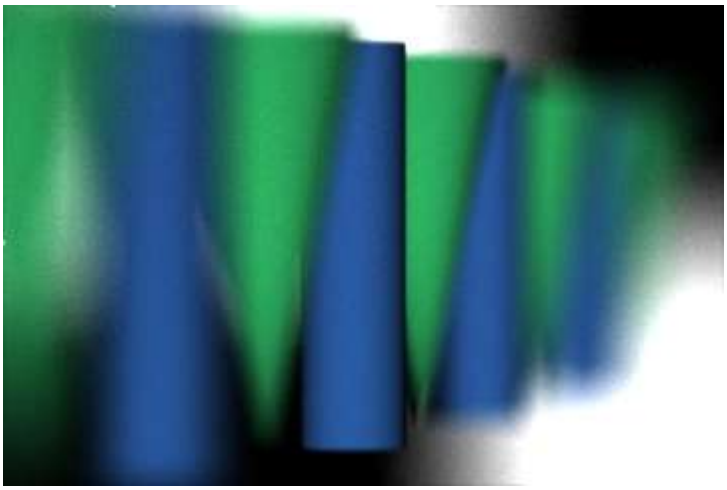
The last thing to affect motion blur is the direction of movement of the shutter in relation to the direction the object is moving in. If the object is moving in the opposite direction to the shutter, its recorded image will appear compressed. Similarly, if it is moving in the same direction as the shutter, it will appear stretched. The faster the object moves, the more it will seem to be compressed or stretched. If the object moves in a direction perpendicular to the movement of the shutter it will appear slanted in its recorded image. The direction of perpendicular movement also affects the direction it slants in (see fig 15). These principals apply to both horizontal shutters (the curtains move horizontally across the film gate) and vertical shutters (the curtains move vertically across the film gate), the only difference being that the compression and stretching effects will be the same type as the shutter i.e. compression will be horizontal if a horizontal shutter is used).

Fig 15: A photo taken with a camera that has a vertical shutter. The shutter starts at the bottom and moves upwards. As the car is moving perpendicular to the shutter (left to right in the image), it makes the car image (and especially the wheel) slant in the direction of its motion. From the way the people in the background are slanted, you can tell the photographer was panning the camera to follow the car. Also, from the minimal blur visible in the image, it is apparent that a fast shutter speed was used. Photograph by Jacques-Henri Lartigues.
(Source: http://members.tripod.com/conduit9SR/Lartigues1.jpg *accessed 7th March 2006)*

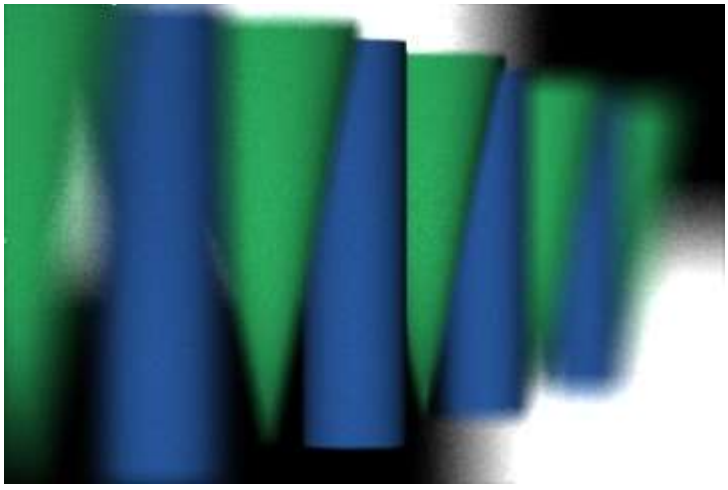
In Maya, time is measured in frames so it makes sense to use them to implement the exposure time of my camera. To get a time exposed image, a number of frames are rendered out, over a frame range that represents a real exposure time, and then added together. Adding the frames together is simulating the additive effect of light on the film plane when it is exposed over a period of time. As most cameras have a minimum shutter speed of 1/1000s, I decided to split the timeframe into 1000 frames per second, thus an exposure of 1/1000s can be achieved with a render equal to one frame. All other exposures will then be a multiple of this e.g. an exposure of 1/500s will be equal to two full frame renders. By increasing the number of frames that can go into a second, the motion blur in long exposure 'photos' will look smoother.

As mentioned earlier, the total time for an exposure is the flash sync plus the shutter speed. This means that a time exposed image won't simply be its shutter speed in frames. To automate the process of having the shutter curtains open and close, and the frames to be rendered, I wrote a mel script that animates the curtains, sets up the render globals in renderman and then renders the appropriate frames (according to the input flash sync and exposure time). See appendix 1. Once these frames are rendered a time exposed image is obtained by adding the single frames together. To do this I wrote a C++ script that reads in the single frames, adds their pixels values together in a buffer then writes the buffer out to a new file. See appendix 2.

6 Final Results

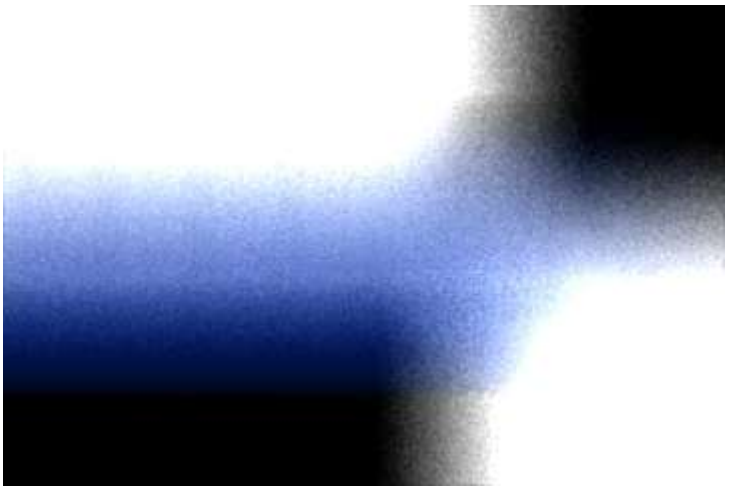


f5.6 1/125 seconds



f8 1/60 seconds

The two renders above have the same exposure value but have used different f/stops and shutter speeds. This was done to illustrate the relationship between aperture and exposure time. The top image has slightly less depth of filed than the bottom one due to the wider aperture. If any of the objects were moving, the bottom render would have more motion blur because of the slower shutter speed. Both images used 1000 sample rays per pixel and took over a day to render.



The render above shows how the camera model simulates motion blur. The blue sphere moves across the frame from left to right.

I decided to render out the frames in a floating point format (openEXR, a format produced by ILM) because in the event of me incorrectly exposing the image, I can just scale the pixel values instead of having to re-render (rendering can take days depending on quality and exposure time). The other thing about exr files is that they scale in terms of exposure. This has the same sort of effect as changing the f/stop or shutter speed but without altering depth of field or motion blur. In other words, it is like changing the iso rating of the film. The iso rating refers to the sensitivity or speed of the film with higher numbers meaning a higher sensitivity (It requires less light to fall on it to produce the same exposure of a slower speed film). It works in a similar way to shutter speed in than 400 iso is a stop up from 200 iso (double the number means double the exposure).

7 Improvements

In the short term, to make the camera more accurate, I'd like to add reflections to the lens shaders to simulate lens flare. I'd also like to add some very faint reflections to the shader applied to the inside of the camera as no material is completely black.

In the current model, the focal length of the lens is fixed, as is the focus. With more lens data, I'd want to add some more lens options and make it possible to interactively zoom the lens. I'd also like to make the focusing interactive using a marker such as a locator to indicate the focal point and make it easier to visualise what the 'photo' will turn out like.

With a few months more time it would be nice to model a fully mechanical camera made to the specifications of an existing camera design, such as an SLR. The user would position the camera to frame the subject and then press down on the shutter release button. The button would set into motion the mechanical sequence that causes the shutter curtains to release (open and close), rather than simply pressing render.

Adding a light meter to the camera would also be very useful, especially as the camera was designed to help users understand how to correctly expose a photo. Light meters work by sampling areas of the proposed photo to obtain their luminance. The meter then suggests an aperture and shutter speed combination that will produce a photo with the sampled area being 18% grey in tone (older cameras suggest an aperture to go with the

current shutter speed setting, known as shutter priority). This would take quite some time to implement as ideally it needs to be fast enough to sample the scene and return a suggestion in real time.

If I was to take the project further from this point, maybe with others helping me, I'd look into properly simulating the lens elements by looking into the way they refract different light wavelengths differently. This would allow the model to correctly simulate chromatic aberration and other aberrations caused by the optics.

It is possible to extend the model even further by looking into the chemistry of film and how it can affect the exposure. The simplest property of film to emulate would be the iso (also known as ASA) rating of the film stock. The film shader could be altered to make it more sensitive i.e. Each ray it casts out would have more effect. This would be the basic implementation of the iso though as higher speed films tend to produce photos with a larger grain size. This is just one aspect of the chemistry of film, and to emulate the full chemistry could take years to research and implement, especially as there are so many different types of film available today.

8 Conclusion

I pleased with the results I obtained from my camera simulation. The model correctly computes depth of field and motion blur and accurately mimics the behaviour of a real camera i.e. Different combinations of f/stop and shutter speed can be used to obtain the same exposure. However, this doesn't mean the simulation is perfect. Apart from the improvements discussed above, I would liked the depth of field to have turned out better. The speckling of the out of focus elements is quite visible. To make the blur seem smoother, more sample rays from the film plane are needed. For my renders I used 1000 rays per pixel as it was the maximum amount that Renderman would allow me to use. If I tried to use more rays the image rendered out black. Similarly, Renderman wouldn't let me use a cone angle greater than 10 degrees as this would also result in a black render (the cone should be a hemisphere so that rays are cast in all directions). My prediction for a render using a wider sample cone is that the blur of out of focus objects would be greater, and therefore make a more realistic result. As I have very little experience with Renderman I wasn't able to rectify this problem. In a way this acted as an optimisation as less rays were needed (a wider cone angle would require more rays to obtain the same results as a smaller cone), but the purpose of my simulation was to get a result as realistic and accurate as possible, not to save render time. The last thing that could have been improved if I had enough time for rendering would be the motion blur. Although it works well, it is possible to see steps in the image where the single frames have recorded the position of the shutter curtain. This is not an error in the model, as it can be improved by increasing the fps (frames per second). This would make the exposures a product of more frames (the setting is currently 1000 fps), but would take an enormous amount of time to render and is therefore unproductive for the purposes of this project.

In the future, as computers become faster and renderers become more efficient, it may be possible to use this camera model interactively or even alter it to perform like a motion picture camera. As it is increasingly important in films to make CG elements seemlessly merge with live footage, an accurate camera model is essential.

9 Acknowledgements

I'd like to thank Ian Stephenson for explaining to me, and helping me understand, the key concepts of the inner workings of 'the camera'.

References

[1] *KOLB C., MITCHELL D., HANRAHAN P. : A realistic camera model for computer graphics. Proceedings of 1995 ACM SIGGRAPH pp 317-324.*

[2] *WARREN J. SMITH , 2005. Modern Lens Design. 2nd edition. New York: McGraw-Hill, pp219 ISBN 0-07-143830-0*

[3] *PIXAR. Renderman Documentation for Proserver version 12.5. California: PIXAR. http://www.pixar.com/*

[4] *WIKIPEDIA ONLINE ENCYLOPEDIA. Bokeh. http://en.wikipedia.org/wiki/Bokeh [Accessed 4th March, 2006]*

[5] *Brown B. , 2002. Cinematography: Theory and Practice. USA: Focal Press pp104-126, 178-192*

[6] *HEIDRICH W, SLUSALLEK P, SEIDEL H-P : An Image-Based Model for Realistic Lens Systems in Interactive Computer Graphics.* PROC **GRAPHICS** INTERFACE. pp. 68-75. 1997.

[7] *GEIGEL J., MUSGRAVE F K. : A Model for Simulating the Photographic Development Process on Digital Images. Proceedings of 1997 ACM SIGGRAPH*

[8] *OpenEXR. Industrial Light and Magic. http://www.openexr.com/ [Accessed 2nd March]*

Appendix 1

```
/*Camera set-up script*/

float $timedivision = (`playbackOptions -q -ps`) *25;        /*Amount of frames per second (for exposure purposes)*/
float $exp = `getAttr Camera.Exposure`;                      /*Exposure time (in seconds)*/
float $flashsync = `getAttr Camera.FlashSync`;               /*Time it takes for each shutter to open/close (in seconds)*/


float $rear = $exp * $timedivision;                          /*Exposure time in frames*/
float $shutter = $flashsync * $timedivision;                 /*Shutter open/close speed in frames*/

float $end = $rear + $shutter;                               /*Frame that the back shutter curtain finishes closing*/

/*Delete all previous keyframes on the shutter curtains*/
CBdeleteConnection "Camera.Front_curtain";
CBdeleteConnection "Camera.Back_curtain";


setKeyframe -v 0 -at Front_curtain -t 0 Camera;              /*Frame that the front shutter curtain starts to open*/
setKeyframe -v 1 -at Front_curtain -t $shutter Camera;       /*Frame that the front shutter curtain finishes opening*/
setKeyframe -v 0 -at Back_curtain -t $rear Camera;           /*Frame that the back shutter curtain starts to close*/
setKeyframe -v 1 -at Back_curtain -t $end Camera;            /*Frame that the back shutter curtain finishes closing*/

/*Set renderman render globals*/

mtor control setvalue -rg computeStart -value 1;             /*Sets start frame for rendering*/
mtor control setvalue -rg computeStop -value ($end-1);       /*Sets end frame for rendering*/
mtor control setvalue -rg animFPS -value $timedivision;      /*Sets frames per second for rendering*/
mtor control setvalue -rg dspyServer -value openexr;         /*Sets file type to exr*/

/*Set off render*/

mtor control renderspool;
```

Appendix 2

```
#include <half.h>
#include <ImfRgba.h>
#include <ImfRgbaFile.h>
#include <ImfArray.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <sys/stat.h>

using namespace std;
using namespace Imf;
using namespace Imath;

/*This function pads the input intger to 4 places*/
string pad4(int i)
{
        string pad;
        ostringstream s;
        s<<i;

        if(i>=1000) {
                pad = s.str();
                return pad;
        }
        if(i>=100) {
                pad = "0";
                pad += s.str();
```

```
                return pad;
        }
        if(i>=10) {
                pad = "00";
                pad += s.str();
                return pad;
        }
        if(i<10) {
                pad = "000";
                pad += s.str();
                return pad;
        }
}

/*This function reads the input exr file and puts the pixel data into the pixels array*/
void readRgba (const char* fileName, Array2D<Rgba> &pixels, int &width, int &height)
{
        RgbaInputFile file (fileName);
        Box2i dw = file.dataWindow();
        width = dw.max.x - dw.min.x + 1;
        height = dw.max.y - dw.min.y + 1;

        pixels.resizeErase (height, width);
        file.setFrameBuffer (&pixels[0][0] - dw.min.x - dw.min.y * width, 1, width);
        file.readPixels (dw.min.y, dw.max.y);
}

/*This function writes the pixel data from the writepixels array into an exr file*/
void writeRgba (const char* fileName, const Rgba *pixels, int width, int height)
{
        RgbaOutputFile file (fileName, width, height, WRITE_RGBA);
        file.setFrameBuffer (pixels, 1, width);
        file.writePixels (height);
}

/*This function loops through the pixel data held within the readpixels array and adds it to the writepixels array*/
void addPixels(Array2D<Rgba> &readpixels, Array2D<Rgba> &writepixels, int width, int height)
{
        for(int i = 0; i < height; i++)
        {
                for(int j = 0; j < width; j++)
                {
                        writepixels[i][j].r += readpixels[i][j].r;
                        writepixels[i][j].g += readpixels[i][j].g;
                        writepixels[i][j].b += readpixels[i][j].b;
                        writepixels[i][j].a += readpixels[i][j].a;
                }
        }
}

/*The main function loops through all the files in the current directory and adds the exr files together that have the correct prefix.  It then
writes this total to a new exr file and opens it in exrdisplay*/
int main()
{
        int width = 360;
        int height = 240;

        string readfilename;
        string writefilename;
        string add;

        string systemcall;

        ifstream fileExists;

        Array2D<Rgba> readpixels(height, width);
```

```
	Array2D<Rgba> writepixels(height, width);


	cout<<"Type in the name of the output file (include the extension .exr)"<<endl;
	cin>>writefilename;

	for(int i = 1; ; i++)
	{

		add = pad4(i);

		readfilename = "frame.";
		readfilename += add;
		readfilename += ".exr";

		struct stat buffer;
		int result = stat(readfilename.c_str(), &buffer);
		if(result < 0) {
			break;
		}

		cout<<"Adding image: "<<readfilename<<endl;

		readRgba(readfilename.c_str(), readpixels, width, height);

		if(i==1)
		{
			writepixels.resizeErase (height, width);
		}

		addPixels(readpixels, writepixels, width, height);

	}

	writeRgba (writefilename.c_str(), &writepixels[0][0], width, height);

	systemcall = "exrdisplay ";
	systemcall += getenv("PWD");
	systemcall += "/";
	systemcall += writefilename;

	system(systemcall.c_str());

	return 0;
}
```