Innovations Report
Georgios Cherouveim

The unknown nature of light.

Introduction

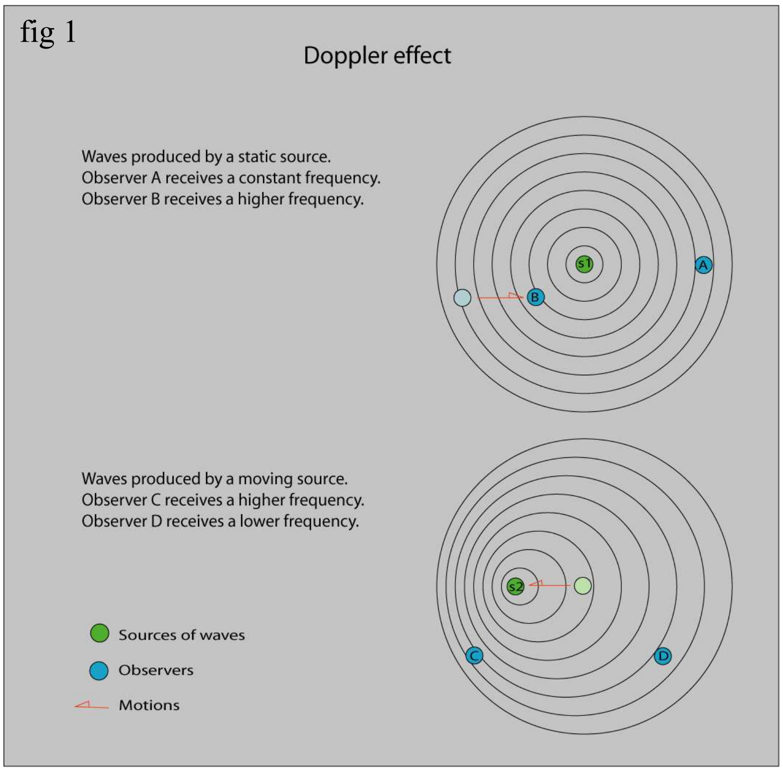
The purpose of this project is to study how light behaves in different time and space scales, as well as in frequencies of the spectrum that are not visible to the human eye. There are optical phenomena, such as Doppler shift, which humans can not easily observe because of the extremely high speed of light. But as the technology of computer graphics has evolved over the last few decades, it is possible to simulate and visualise some of those phenomena. In this report, the practical techniques and the theory needed to simulate such phenomena in computer graphics is presented. In one case, a detailed implementation is given for the Doppler shift. On the other hand, a theoretical discussion explains the advantages and disadvantages of different ways to implement simulations like bending of light near strong gravitational fields and also the delay of visual perception because of the finite speed of light.

Doppler shift

The Doppler shift is the phenomenon when the pitch of a wave frequency changes because of the relative speed between the source and the observer. This can take place to any kind of waves and pulses, from sea waves, to the sounds and the electromagnetic radiation including the visible light. It is named after the Australian mathematician Christian Andreas Doppler who was the first one who to observe it. (*ref 1*)

If a source emits a frequency with a fixed length A, but at the same time it moves toward the observer with a speed of A per frequency period, the observer will perceive the wave as having double the frequency. If the light emits in the opposite direction and away from the observer, the perceived wave will have half the frequency. The same applies if the source is static but the observer is moving, or even if both are moving. All that matters is their relative speed. So in general if the source and the observer are getting closer, the frequency increases and if they are moving away the frequency decreases. The faster the wave travels, the less noticeable the shift is and vice versa. (*fig 1*)

fig 1


Doppler effect
Waves produced by a static source.
Observer A receives a constant frequency.
Observer B receives a higher frequency.
S1
A
B
Waves produced by a moving source.
Observer C receives a higher frequency.
Observer D receives a lower frequency.
S2
C
D
Sources of waves
Observers
Motions

The easiest way to perceive this shift is with sound. This is because it moves relatively slow. The common example is with the ambulance's horn as it drives toward us and then away from us. We first listen the horn sound in a higher pitch, then as it drives next to us we notice a change of the shift for a while and then the sound's pitch becomes much lower. On the other hand the driver of the ambulance hears the sound as it is produced from the speaker.

It gets more interesting when this phenomenon is applied to visible light. The change of frequency in visible light means change in colour, because each colour is just a different frequency of the visible electromagnetic spectrum. This is much harder to spot because light travels really fast for the human time and space scale. Even though, this phenomenon is widely used by the astronomers to calculate the distance of stars and other objects in space which are really far away and the classic method of trigonometry using the different parallax cannot be applied. As the Universe expands, the further a star is, the faster it moves away from us. So if the star emits a known radio wave spectrum signature, the astronomers can measure the red shift and from that they can calculate its distance. Another interesting aspect of this phenomenon is that invisible frequencies of radio wave spectrum can become visible with the proper shift. So very distant objects like quasars, will show part of their ultraviolet radiation as visible to the human eye colours.

Vesto Slipher in 1912 first used the Doppler shift in astronomy. Then in 1929 Edwin Hubble and Milton Humanson, after using the same method, they discovered that the Universe expands.

Mental Ray custom shader

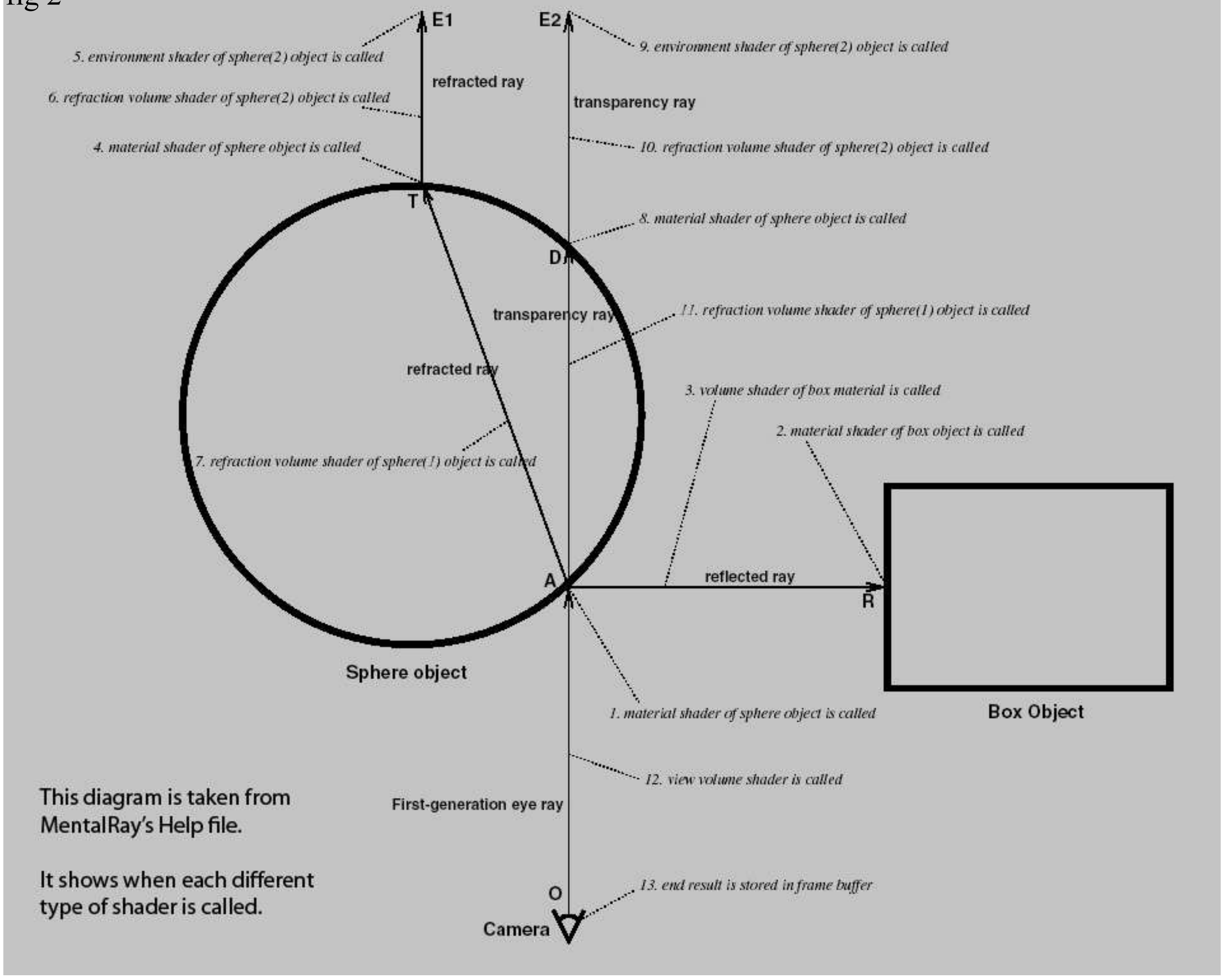
We decided that the Mental Ray renderer would be used to implement the shader which simulates the Doppler shift. Writing shaders for Mental Ray was not as easy as in Renderman, because of the poor documentation. Mental Ray was chosen, because it is better integrated inside Maya which means it is easier to use as a shader. A couple of weeks were spent, compiling and linking a basic shader. Even though most of the information needed can be found in the main Mental Ray help file, this is scattered in over 400 pages and in illogical order (*ref 2*). In the end, we decided that a small tutorial could be written in order to better explain the whole process. This tutorial is now online (*ref 3*) and it has already been used by other people who wanted to start in implementing shaders.

Doppler shader

From the start the idea was to find a way to make the shader generic and be able to apply it to a whole scene at once without having to change the shaders of each object. After some reading and tests, the author $we decided it had to be a volume shader which can be attached to the camera and alter the look of any given scene without any extra work. This is because, this type of shader is called after all the other regular shaders which means the frame buffer already has a colour which Doppler will alter according to the shift.(*fig 2*)

The next concern was how to get the velocity information inside the algorithm. Luckily this information is available in MentalRay by default, as it is the way the renderer applies motion blur to the scene in a post render stage. There are two algorithms for motion blur within MentalRay. The first one is 'Linear' and can detect translations on a per object basis, when the second one is 'Exact' which works on sampled point basis and means it can work for deformable objects as well. This information is a motion vector which shows the direction and the magnitude for each point sampled relative to the camera. MentalRay also gives information about the coordinates of the point sampled which can then be converted to either world or camera space.

fig 2



E1
E2
5. environment shader of sphere(2) object is called
9. environment shader of sphere(2) object is called
refracted ray
6. refraction volume shader of sphere(2) object is called
transparency ray
4. material shader of sphere object is called
10. refraction volume shader of sphere(2) object is called
T
8. material shader of sphere object is called
D
transparency ray
11. refraction volume shader of sphere(1) object is called
refracted ray
3. volume shader of box material is called
2. material shader of box object is called
7. refraction volume shader of sphere(1) object is called
A
reflected ray
R
Sphere object
1. material shader of sphere object is called
Box Object
12. view volume shader is called
This diagram is taken from MentalRay's Help file.
It shows when each different type of shader is called.
First-generation eye ray
O
13. end result is stored in frame buffer
Camera

The relative speed of the point to the camera is calculated by measuring the distance of the point to the camera, then using the motion vector to calculate the point's position in the next frame. The distance to this new position is then measured and the relative speed is the difference between these two distances. If the difference is zero, the point is static. If it is positive, it means it travels away from the camera and vice versa. Now this variable is able to drive the shift of the colour.

In the early stages of the shader implementation, red or blue colour was just added to the point sampled to represent the relative speed from the viewing camera. But after that, a proper algorithm had to be written which would do the actual colour shift.

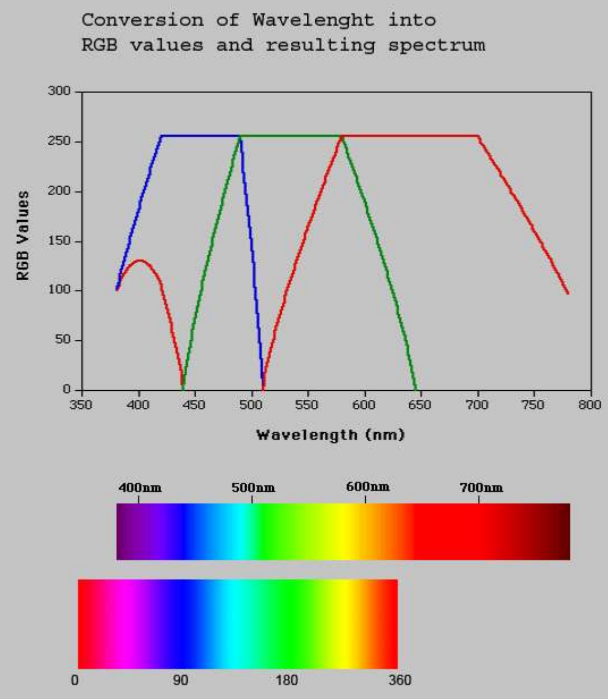
In the first place an accurate approach was researched which was ways to convert the RGB values into a frequency equivalent, do the shift with the real Doppler formula and then convert it back to RGB. After a lot of experiments, it was realised that this was not that easy but more important not that accurate. An algorithm which converts light Wavelengths to RGB value was fount, but not the inverse (*ref 4*). RGB is a 3d value and has more information than a colour wavelength which is one dimensional. RGB has a greater range of colours as it includes information like the saturation and the lightness. Even if the implementation had taken that way, by working out an algorithm which would convert RGB to Wavelength, the process would be much more complicated and it would probably have many accuracy errors. A source of this inaccuracy is the fact that RGB cannot reproduce the whole visible spectrum exactly. There are some RGB values which correspond to a small range of wavelengths. In this case the algorithm would be:

1. Temporary convert the RGB to HSV.
2. Store the saturation and lightness of the current colour.
3. Push the RGB to the maximum saturation of the colour.
4. Convert RGB to light wavelength.
5. Shift the wavelength.
6. Convert the wavelength back to RGB.
7. Restore the saturation and lightness to the final colour.

On the other hand it was noticed that the colour range the Wavelength to RGB conversion algorithm produces is almost identical with the hue scale (*fig 3*). The main difference is that the hue scale starts and finishes with red, whereas in reality the visible spectrum goes from red to violet and then fades to black as it gets in to the invisible part. This can be expressed as the following algorithm:

1. Convert the RGB to HSV.
2. Apply the shift by changing the Hue value.
3. Fade the colour to black if it is near the boundaries of the visible spectrum.
4. Convert the new HSV to RGB.

fig 3



The HUE range that almost matches the result of the wavelength convesrion algorithm

Even if this approach does not involve the real Doppler shift formula, it is much more accurate and faster than the first one as it involves fewer conversions. It was quite easy to locate algorithms to convert colour between those two different models (*ref 5*), so two functions were implemented. The next issue was how much to shift the hue value for a given relative speed so as to approximate the Doppler shift. The shift itself is not only dependant on the relative speed but also on the speed of light. In reality the speed of light is constant but in the case of the shader a much smaller value must be used for the speed of light, so the effect can be visible for smaller scales. It was decided the speed of light should be a variable, which can be changed by the user $so to better control the effect.

$z = (rs / c) + 1$

The shift (z) is equal to the ratio of the relative speed (rs) over the speed of light (c) plus one (*ref 6*). Afterwards, it is this (z) factor which multiplies the colour frequency. So if the camera is moving towards the object with the speed of light, the frequency of the colour doubles. What needed to be done, was to map the frequencies in to the hue scale. The visible light spectrum starts from a wavelength of about 400nm and goes up to 700nm. So at 400nm there is violet colour with a hue of 293 and on the other side is orange at 600nm which has hue 37. It is obvious that the relationship is such that doubling the frequency equates to a shift in hue of approximately 512 units. The final formula is:

$HueShift = (relativeSpeed / lightSpeed) * 512$

```
H += relSpeed / (speedOfLight / frameRate) * 512;
```

Light speed is first divided by the playback frame rate because the relative speed is measured on a per frame basis.

The next step was to darken the colour as it shifts out of the visible spectrum. It was mentioned previously that the RGB scale has some colours which do not correspond to the visible light spectrum. These correspond to the hue values greater than 300. For values larger than 300 the hue is set to 300 and the intensity drops off linearly over the range 300 to 360.

```
if( H>300 )
{
      if( H>360 )
            H = 360;

      V *= 1-( (H-300)/60 );
      H=300;
}
```

The same is true for the other side of the spectrum: the colour attenuates if the hue drops below 0.

```
if( H<0 )
{
      if( H<-60 )
            H = -60;

      V *= (H+60) / 60;
      H=0;
}
```

Disadvantages of the Doppler shader

The main problem with the shader as it works right now in its current state is that even if there is no shift, the colours with hue greater than 300 are lost, as it is obvious from the above algorithm. The problem is that these colours are combinations of more than one electromagnetic wavelength so they must first be broken down, shifted and then recomposed. Unfortunately this was beyond the scope of this project, so it was not researched.

There is also another drawback of the shader which makes it not work in a realistic way. As was mentioned in the explanation of the phenomenon, for great shifts invisible radiation like infra-red, ultraviolet or even microwaves and X-rays can become visible to the human eye. This cannot be simulated with this shader as there is no information about the invisible radiation in a common 3D application's scene so it simply turns the colours to black.

A solution to this problem would involve a whole collection of new shaders and new colour channels in the frame buffer. Surface shaders which would be able to calculate how the object diffuses, reflects or refracts the different types of radiation. The surface could also emit its own radiation, for example if it is hot. Light shaders which would light the scene in the invisible spectrum. Also different atmospheric shaders since some radiations react differently. For example infra-red is not blocked by fog, but it has a different refraction index from visible light. The hardest part, which would involve quite a lot of research, would be the representation of the invisible radiation. In computer graphics, three colour channels are used, only for the visible light. Since the visible light is just a small portion of the whole electromagnetic range, many more channels would be needed to store and manipulate the extra information.

Other applications of the shader

After the shader was finalised, it was tested with a few animation renders which illustrate the Doppler phenomenon in various situations. In the mean time though, it was noticed that the shader could produce some interesting results with other more abstract scenes.

In some of the cases though, the fact that the colour was turning black as it was going out of range was not desirable. A simple solution was to loop the hue and not attenuate the colour. So an extra boolean attribute was added to the shader node.

```
if( loopHue )
{
      while( H >= 360 )
            H -= 360;

      while( H < 0 )
            H += 360;
}
```

This gives interesting results with deformable surfaces. One of the test renders was to convert a simple plane with a photo to a soft body. Since different areas of the surface deform at different speeds an interesting effect is produced. It looks like the colour patterns made by oil in to water. On the other hand looping the hue in an ordinary scene where the camera moves around, does not give such an appealing result. The repetition and vast array of colour patterns makes the scene look more like a disco.

Relativistic Rendering

One of the initial ideas behind this project was the research and development of a rendering technique which treats light as having a finite velocity like in reality. Off course, since light travels extremely fast and appears instantaneous to us, all known commercial renderers assume that light has an infinite speed. After some research and lot of thinking it was realised this was extremely ambitious within the scope of this project. It would involve the implementation of a whole new renderer which would work in a completely different way to current scan-line or raytraced renderers. The Doppler shader was part of this idea, but in the end it seemed that it was the only part that could be implemented. So the rest of the idea was kept at a theoretical level, researching some approaches and the benefits of it in computer graphics. There are quite a few papers about this type of rendering, but unfortunately all that it was possible to be locate was their abstracts.

Such a renderer probably would not have any practical application, beyond that of an educative tool to illustrate some phenomena which take place on much greater scales and cannot be perceived by us in our natural world. Of course it would be really interesting to see and explore these visual effects as not many people are familiar with these phenomena. It is also interesting to point out that even though there are so many theories and much research on how light truly reacts in a relativistic way, scientists only have a small collection of visual examples on which to base their theories.

The two main simulations that would be able to run with such a renderer would be the bending of light near strong gravitational fields and the delayed perception of actions which take place far away from the observer.

Bending of Light

In the past some scientists believed that light was a wave, whilst others argued that light was a type of particle, but all of them thought that light could only travel in a straight line inside a vacuum if there was nothing to refract it. Nowadays it is known that light's behaviour varies depending on the occasion and may also bend. It was when Albert Einstein published his paper in 1916 on general relativity that people started to think of light in a different way (*ref 7*). Later, on 1919, this was proved when astronomers noticed a slight bend of the light near the Sun where the gravity is quite strong. The angular positions of known stars were a bit different as the line of sight approached the Sun. Nowadays scientists have discovered this phenomenon on a greater scale near massive black holes which have enormous gravity. Those create a gravitational lens effect which means some objects which appear next to the black whole may have a double image.(*fig 4*) It is even more interesting if we imagine how things would look if we were able to stand close to a black hole. As we approach the event horizon of the black hole, the light bends more and more and falls almost vertical to the centre of the hole. Because of that we would not be able to see any light coming from our surroundings but only from a small circular window above us. Within this small window, we would not only be able to see a view of the entire universe, but also many duplicates of it because some photons can orbit a few times around the hole before they reach our yes.(*fig 5*)

fig 4

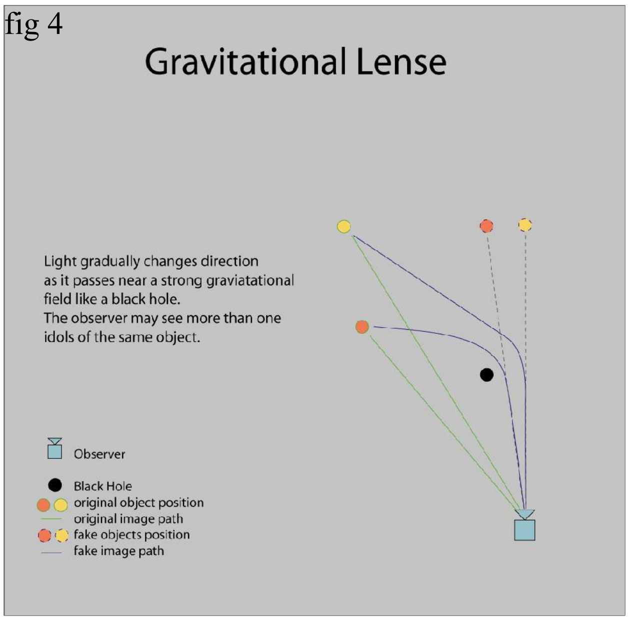
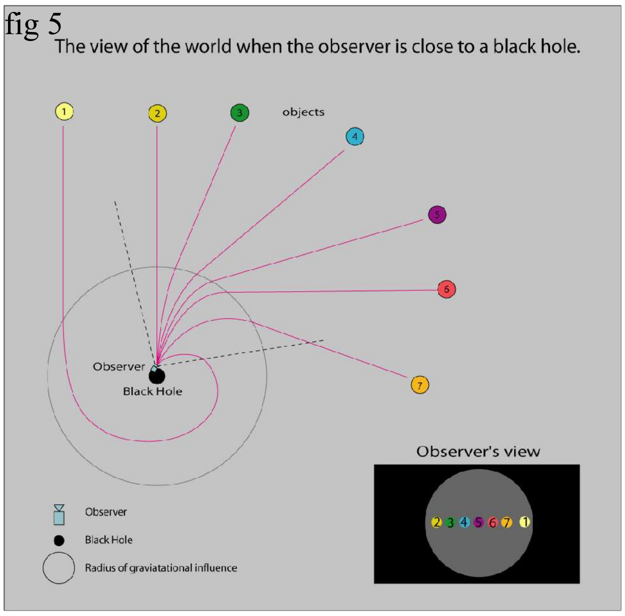

Gravitational Lense
Light gradually changes direction as it passes near a strong graviatational field like a black hole.
The observer may see more than one idols of the same object.
Observer
Black Hole
original object position
original image path
fake objects position
fake image path

fig 5


The view of the world when the observer is close to a black hole.
objects
1
2
3
4
5
6
7
Observer
Black Hole
Observer's view
2 3 4 5 6 7 1
Observer
Black Hole
Radius of graviatational influence

The easiest way to understand how black holes can trap light and also bend it is if we first consider the escape velocity of a celestial body. The escape velocity is the minimum velocity an object must have in order to escape the gravitational field of a specific planet or star. On Earth, if we throw a rock vertically, we expect it to fall back down because of gravity. But if we throw it with a speed greater than 11.1 km/sec, which is Earth's escape velocity (ignoring air friction), the rock will not fall back.

$$v = sqrt(2GM/R)$$

(v = escape velocity, G = gravitational constant, M = mass of the object, R = radius of the object)

Given the formula which describes the escape velocity we notice that the greater the mass of the celestial object or the smaller its radius, the greater the escape velocity. So with celestial bodies like black holes which have a massive mass but also a tiny radius, the escape velocity exceeds the speed of light within the event horizon. This means, that even if an object reaches the speed of light, it will not be able to escape the gravitational force of a black hole. Of course, the same happens with the light itself and this is the reason why anything that falls into a black whole, cannot escape.

Also, when light is influenced by a big gravitational field, it is also affected by the Doppler effect as well. For example the light from a star with an escape velocity of half the speed of light, would have double the wavelength (or half the frequency) of the light emitted. Or if we observe an object falling into a black hole, we will see its colour changing through the spectrum, before it disappears entirely.

Delay of visual perception

Since light has a specific speed visual information will take some time to reach our eyes. This fact is not that obvious to us. We can compare this occurrence to sound which also has a finite speed. For example, we see lightning before we hear the thunder. We see a distant firework before we hear it explode. In the case of sound the difference is that we have a frame of reference: a visual event that reaches us at nearly the same moment the event took place. This means that we can easily measure the delay between the event and the aural evidence of that event. On the other hand, this is not possible with light because there is no faster way to get the information that an event took place earlier than to perceive it visually. It is like a blind man who cannot perceive the delay in sound, because he cannot see when the event happened. So for example if there was a major change on the surface of the Sun, we would only be able to see it about eight minutes later. Even if we had a remote camera in orbit close to the Sun, we would not be able to learn about this event earlier, because radio waves travel at the same speed as light.

The perception we have about the environment is heavily based on our vision. We believe things we see and we may doubt others that we have not seen with our own eyes. We take it for granted that everything we see happens at the present time. This is not true. We actually see the light which left an object a little bit earlier, so the further away we look, the further back in time we see. We look at the Sun, but we are actually observing how the Sun was 8 minutes before. This is more than common in astronomy. Because of the vast scale of space, everything we see in the sky, has taken place ages ago. The closest star to us, Alpha Centauri, is 4.35 light-years away, which means we see it as it was about four years ago. Going even further, we see the closest galaxy, Andromeda, as it was 2.9 million years ago when our planet was ruled by dinosaurs. Going to the extreme, the furthest celestial objects that astronomers can see are Quasars which are more than 10 billion light-years away. If we think that the age of the universe is estimated to be about 14 billion years, we understand that Quasars do not exist any more, but they are actually evidences of how the universe was in its prime stages, in a different era.

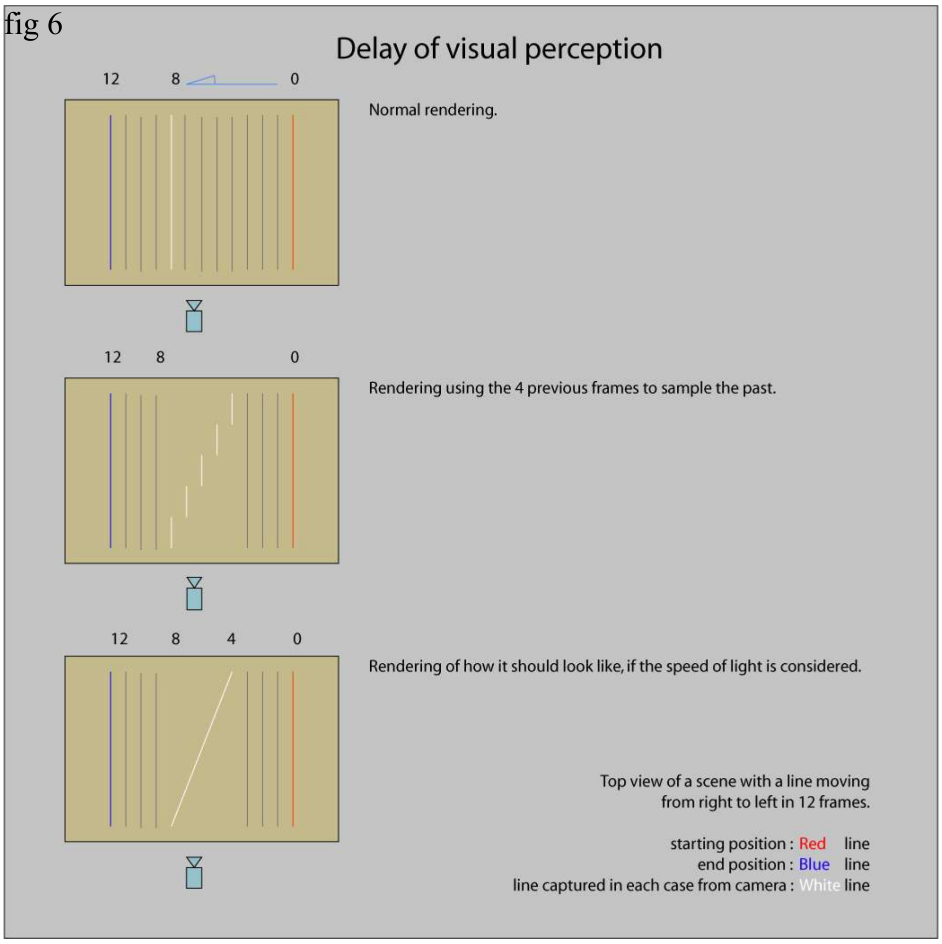
There is an interesting artefact of this phenomenon which is not possible to observe, but would be nice to simulate through the use of computer graphics. If we had a space ship which could travel at any speed and also a very powerful telescope with which we could observe a specific event on Earth, we would be able to see the event taking place at any speed, even backwards. Travelling away from the Earth with half the speed of light and looking backwards, we would see the event evolving at half of its speed. When we reach the speed of light, the event will freeze as new photons will not be able to reach our eyes. And while travelling with double the speed of light, we would be able to see the event evolving in reverse. Of course in reality this is not possible either because nothing can travel faster than light (as we at least know until now), or because of the huge Doppler shifts. Also travelling faster than light, would create a shadow behind us, which would not make it possible to see anything backwards.

Theoretic approach for a Relativistic Renderer

Possibly such a renderer would be able to work with raytracing. Rays are emitted from the camera for each sample of each pixel and they travel in a straight line until they hit an object and return its colour. But because we need to make the ray hit an object in a previous time, the renderer must be able to refer to different ‘frame-scenes’ (renderers are typically not aware of animation, they just render different static scenes for each frame. Only in the case of motion blur does the renderer need information about the animation of the objects in the scene). So for every frame duration that the ray travels without hitting any objects, the renderer should change ‘frame-scene’ whilst keeping the same attributes for the ray (world space position, direction, velocity and colour).

The drawbacks of this approach would first of all be efficiency. More than one ‘frame-scene’ must be loaded into memory so the renderer can refer to them. Moreover, going frame by frame backwards to resolve where the ray is going to hit, would be just an approximation. Normally in animation we sample each second with 25 static frames, whereas in this case ideally the samples must be infinite to get the proper output. For example think of a cylinder which extends parallel to the Z axis of the camera and is moving sideways. With an infinite number of samples, the cylinder would look sheared. But using previous ‘frame-scene’ would result in a chopped cylinder with many straight fragments (*fig 6*). Also if an object is moving quite fast, it will not block the rays which happened to travel in between the object's current and previous position. A solution to this problem may be an algorithm which takes all the moving objects of the scene and makes a swept collision test with each ray. So the point at which the first collision takes place is the point the ray should render. Of course the renderer should be aware of all animation within the scene. Again something like this would be quite slow, as collision test algorithms are generally quite expensive in terms of processing power.

fig 6

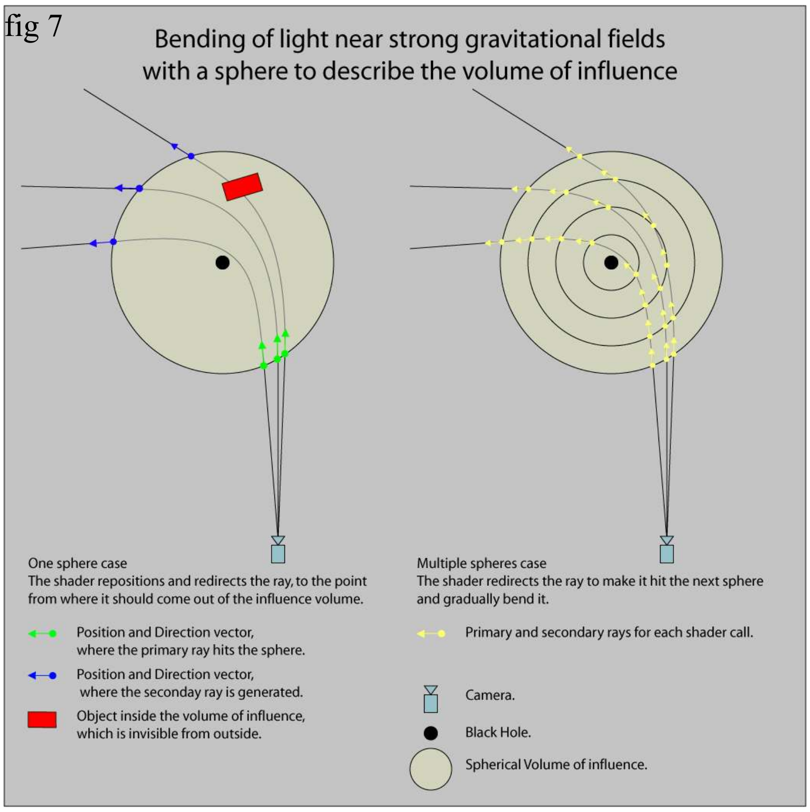

Delay of visual perception
12
8
0
Normal rendering.
12
8
0
Rendering using the 4 previous frames to sample the past.
12
8
4
0
Rendering of how it should look like, if the speed of light is considered.
Top view of a scene with a line moving from right to left in 12 frames.
starting position : Red line
end position : Blue line
line captured in each case from camera : White line

For the case of bending light, the ray must be considered as a particle. While the collision algorithm runs, it should also evaluate the dynamics for each gravitational field which may alter the ray's direction. One of the initial ideas behind this project, was the implementation of a refractive shader which would be able to simulate this bending. After some attempts, the idea was rejected. The first reason was that the shader should lie on a surface, which in this case would be a sphere that would define the volume of influence of the black hole. But gravity does not have a fixed boundary, but attenuates gradually ($g = 1/d^2$).

In addition the rays cannot be bent but they can only be redirected in each shader call. This leads to two different approaches: One approach would be a shader which can redirects and repositions the ray, from where it first hit the influence sphere, to where it should come out. The problem with this is that any object inside the influence sphere would not be visible from outside, and also the shader would not work properly if the camera is placed inside the sphere.

The other approach would be to use multiple spheres with the shader, where each one gradually redirects the ray. The overall result might be correct, but it would only be an approximation. The more spheres to describe the gravitational field of the black hole, the more accurate the result, but more time consuming in terms of rendering as there would be more function calls. (*fig 7*)

fig 7


Bending of light near strong gravitational fields
with a sphere to describe the volume of influence
One sphere case
The shader repositions and redirects the ray, to the point from where it should come out of the influence volume.
Multiple spheres case
The shader redirects the ray to make it hit the next sphere and gradually bend it.
Position and Direction vector, where the primary ray hits the sphere.
Position and Direction vector, where the seconday ray is generated.
Object inside the volume of influence, which is invisible from outside.
Primary and secondary rays for each shader call.
Camera.
Black Hole.
Spherical Volume of influence.

Conclusion

After this short period of research, it is obvious that all those phenomena can be simulated in one way or another with the use of computer graphics. Their application, however, would probably only be useful for educational purposes, to illustrate how light behaves in scales different to those that we are used to. Even if an effect like this is required for a film or similar, it would probably be better to approach the task in a more traditional way as the director would have better control over the final result.

As regards the personal benefits gained from this project, the most prominent is the fact that the subject matter was of great interest to me. I have tried to verify the information in this report to the best of my ability. However, much of the information used originates from books studied in the past (to which I no longer have access). On the other hand, the process of writing this report has lead to an even greater understanding of the subject matter. This project has also given me the opportunity to explore shader writing, which I was hoping to do during the period of this degree. Besides the shader written for this project, I have already implemented a couple more that I am currently using in my other two projects.

Finally, I would like thank my supervisor Ari Sarafopoulo, for his guidance on this project. My classmate Sander Van Der Steen for his help with Mental Ray and my flatmate Adam Cubbit for reading and correcting this report.

References

ref.1 *J J O'Connor* and *E F Robertson.* 1998 [online] School of Mathematics and Statistics, University of St. Andrews, Scotland. [Accessed February 2005] http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/Doppler.html

ref 2 Anon. 2004 Mental images Gmbh. Mental Ray user manual.

ref 3 Cherouveim, G. 2005 [online]. Mental Ray tutorial. http://www.ch3.gr/tutorials/mentalRay/

ref 4 Bruton, D. [no date] [online] Colour Science [Accessed February 2005] http://www.physics.sfasu.edu/astro/colour/spectra.html

ref 5 Cardani, D. 2001 Adventures in HSV Space [online] [Accessed March 2005] http://www.buena.com/articles/hsvspace.pdf

ref 6 Edward L. 2002 [online] Doppler Shift explanation [Accessed March 2005] http://www.astro.ucla.edu/~wright/doppler.htm

ref 7 Nobel Lectures. Physics 1967. Elsevier Publishing Company Amsterdam. [online] [Accessed March 2005] http://nobelprize.org/physics/laureates/1921/einstein-bio.html

Books

George B.Field - Eric J.Chaisson 1987. The Invisible Universe
Vintage Books USA

Kip S.Thorne 1994. Black Holes and Time wraps
W.W. Norton & Company Ltd.

Stephen Hawking 2002. The Universe in a Nutshell
Bantam Press

Papers

Wide-band relativistic doppler effect visualization
Hsiung, Ping-Kang (Carnegie Mellon Univ, Pittsburgh, PA, USA); Thibadeau, Robert H.; Cox, Christopher B.; Dunn, Robert H. P.; Wu, Michael; Olbrich, Paul Andrew **Source:** *Proc First 90 IEEE Conf Visualization Visualization 90*, 1990, p 83-92, 465-

Spacetime visualization of relativistic effects
Hsiung, Ping-Kang (Carnegie Mellon Univ); Thibadeau, Robert H. **Source:** *ACM Eighteenth Annual Computer Science Conference (CSC90)*, 1990, p 236-243

Real-world relativity: Image-based special relativistic visualization
Weiskopf, D. (Inst. for Astronomy and Astrophysics, Section Theoretical Astrophysics, University of Tu&die;bingen); Kobras, D.; Ruder, H. **Source:** *Proceedings of the IEEE Visualization Conference*, 2000, p 303-310+571

Appendices

```
#include "shader.h"

/************************/
/* RGB to HSV to RGB  */
/************************/
float min( float a, float b, float c)
{
	if( a<b && a<c )
		return a;
	else
	if( b<c )
		return b;
	else
		return c;
}

float max( float a, float b, float c)
{
	if( a>b && a>c )
		return a;
	else
	if( b>c )
		return b;
	else
		return c;
}

// r,g,b values are from 0 to 1
// h = [0,360], s = [0,1], v = [0,1]
//		if s == 0, then h = -1 (undefined)

void RGBtoHSV( float r, float g, float b, float *h, float *s, float *v )
{
	float mini, maxi, delta;

	mini = min( r, g, b );
	maxi = max( r, g, b );
	*v = maxi;				// v

	delta = maxi - mini;
	if( delta == 0 )
		delta = 0.001;

	if( maxi != 0 )
		*s = delta / maxi;		// s
	else {
		// r = g = b = 0		// s = 0, v is undefined
		*s = 0;
		*h = -1;
		return;
	}

	if( r == maxi )
		*h = ( g - b ) / delta;		// between yellow & magenta
	else if( g == maxi )
		*h = 2 + ( b - r ) / delta;	// between cyan & yellow
	else
		*h = 4 + ( r - g ) / delta;	// between magenta & cyan

	*h *= 60;				// degrees
	if( *h < 0 )
		*h += 360;

}

void HSVtoRGB( float *r, float *g, float *b, float h, float s, float v )
{
	int i;
	float f, p, q, t;

	if( s == 0 ) {
		// achromatic (grey)
		*r = *g = *b = v;
		return;
	}
	h /= 60;				// sector 0 to 5
	i = floor( h );
	f = h - i;				// speedOfLightial part of h
	p = v * ( 1 - s );
	q = v * ( 1 - s * f );
	t = v * ( 1 - s * ( 1 - f ) );
	switch( i ) {
		case 0:
			*r = v;
			*g = t;
			*b = p;
			break;
		case 1:
			*r = q;
			*g = v;
			*b = p;
			break;
		case 2:
			*r = p;
			*g = v;
			*b = t;
			break;
		case 3:
			*r = p;
			*g = q;
			*b = v;
			break;
		case 4:
			*r = t;
			*g = p;
			*b = v;
			break;
		default:		// case 5:
			*r = v;
			*g = p;
			*b = q;
			break;
	}

}

/*********/

struct doppler
{
	miInteger frameRate;
	miScalar speedOfLight;
	miBoolean loopHue;
};



/*********/
/*********/


DLLEXPORT int doppler_version(void) {return 1;}

DLLEXPORT miBoolean doppler(miColour *result, miState *state, struct doppler *param)
{

	miInteger frameRate;
// define the frame rate of play back to determine the speed of light on per frame basis
	miScalar speedOfLight;
// define the speed of light in units per second
	miBoolean loopHue;
// loops the Hue scale, and doesnt attenutate the colour.


	miVector pos0;
	miVector pos1;
	miScalar dist0;
	miScalar dist1;

	miVector motion;
	miScalar relSpeed;

	float R, G, B;
	float H, S, V;


	mi_point_to_camera( state, &pos0, &state->point );
// set position of the current frame
	mi_vector_to_camera( state, &motion, &state->motion );
// set motion
	mi_vector_mul( &motion, 2.5 );
// motion per frame

	mi_vector_add( &pos1, &pos0, &motion );
// position of the next frame
```

```
       dist0 = mi_vector_norm( &pos0 );
// distance of the current frame
       dist1 = mi_vector_norm( &pos1 );
// distance of the next frame

       relSpeed = dist0 - dist1;
// relative speed units/frame ( positive:blue  negative:red )



       if (state->type == miRAY_LIGHT)
              return(miTRUE);


       frameRate = *mi_eval_integer(&param->frameRate);
// take parameter from node's attribute
       speedOfLight = *mi_eval_scalar(&param->speedOfLight);
// take parameter from node's attribute
       loopHue = *mi_eval_boolean(&param->loopHue);
// take parameter from node's attribute



       R = result->r;
       G = result->g;
       B = result->b;



       RGBtoHSV( R, G, B, &H, &S, &V );
// Convert RGB to HSV

       H += relSpeed / (speedOfLight / frameRate) * 512;
// Shift Hue


       if( loopHue )
// what to do for the values outside the range (0-360)
       {
              while( H >= 360 )
                     H -= 360;

              while( H < 0 )
                     H += 360;
       }
       else
       {
              if( H>300 )
// if new Hue value is not going to be visible, fix it and just darken it
              {
                     if( H>360 )
                            H = 360;

                     V *= 1-( (H-300)/60 );
// How much to be darkened
                     H=300;
              }

              if( H<0 )
// if new Hue value is not going to be visible, fix it and just darken it
              {
                     if( H<-60 )
                            H = -60;

                     V *= (H+60) / 60;
// How much to be darkened
                     H=0;
              }
       }


       HSVtoRGB( &R, &G, &B, H, S, V );
// Convert colour back to RGB

       result->r = R;
       result->g = G;
       result->b = B;
       result->a = result->a;



       return(miTRUE);
}
```