Approximating the Fire Flicker Effect Using Local Dynamic Radiance Maps

Jonathan Brian Metzgar
University of Colorado at Colorado Springs
jonathan@metzgar-research.com

Sudhanshu Kumar Semwal
University of Colorado at Colorado Springs
ssemwal@uccs.edu

ABSTRACT

Realistic fire and the flicker effect is a complicated process to simulate in realtime and little work has been done to simulate this complicated illumination effect in realtime. Fire is not a directionally uniform source of light but varies in intensity not only with time but also with direction. Most realtime applications use a standard point light source model for local illumination effects and may use a model to change the light source intensity with time but not direction. The problem is that point light sources are isotropic, but many sources of light have anisotropic qualities as well. Radiance maps and Precomputed Radiance Transfer (PRT) have been used to increase realism at realtime interactive frame rates. These models approximate global illumination by applying an environment map (typically approximated with spherical harmonics) to get their soft lighting effect. In this paper we present Local Dynamic Radiance Maps (LDRM) which uses radiance maps in a local illumination model to add anisotropic behavior to light sources. We implemented a realtime rendering engine that supports shadow mapping and the physically based Cook-Torrance model to approximate global illumination. In particular, we generate dynamic radiance maps using Perlin noise to simulate the nonlinear radiance of fire and we also implement a rudimentary Lattice-Boltzmann flame rendering effect. Finally, we show how LDRM can be applied not just to approximating the fire flicker effect, but as a general framework for simulating the illumination properties of other nonlinear light sources.

Keywords: radiosity, global illumination, fire, Lattice-Boltzmann, radiance maps, shadow-mapping.

1 INTRODUCTION

Advances in graphics processing units (GPU) have resulted in not only improved speed and quality of computer generated images, but now feature massively parallel processors capable of running several general purpose programs. This parallelism allows for the implementation of global illumination algorithms. Physical simulations of natural phenomena like fire and water are taking advantages of the hardware acceleration.

Rendering fire is a big challenge for computer graphics because it touches so many areas of image generation. It is even harder to do it well in realtime. One would want to eventually render fire based on a full 3D simulation using Navier-Stokes equations and render a scene in realtime using the illumination effects modeled by a radiosity algorithm. Since this is not practical, fire imagery is often created through precomputed renderings, video, or particle effects and the illumination comes from a point light source with a dynamic intensity. But, simply modulating the intensity and location of a point light source does not adequately model the way that fire radiates in a nonlinear way. It is this nonlinear radiance that causes the flicker effect to occur.

For the last decade, radiance maps and precomputed radiance transfer have become essential for creating high quality realtime visualizations. Essentially by utilizing approximations like spherical harmonics, they can quickly apply the illumination model to objects in a scene and get radiosity like shading effects. This model works with an outward-in approach where the incoming light is mapped to a sphere which is mapped to the precomputed radiance map. We propose a variation of this method that 1) dynamically computes the radiance and 2) is a local source of radiance which can move around inside an environment. We call these local dynamic radiance maps or LDRM. The main goal of the LDRM model is to make lights appear and act like they belong in the scene by modeling their anisotropic behavior as a function of time.

In this paper, we present a new way to approximate the flicker effect by procedurally generating a LDRM into a cube map. This LDRM is projected outwards from the position of the fire source through the cube texture and onto the surrounding scene's geometry simulating

the first bounce of radiosity. To achieve our results, we have chosen bump mapping, physically based lighting, and soft shadow mapping algorithms to calculate direct and indirect illumination of the fire source. Finally, we implement a procedurally simulated fire effect based on a Lattice-Boltzmann model which can be simulated on the GPU or CPU to approximate the flames emitted from a torch. This is rendered in our scene at the location of our fire source.

2 PREVIOUS WORK

Simulating fire is a fluid simulation problem that has a variety of solutions ranging from artist derived flame profiles, procedural generation of flames, and fluid simulation of flames. [Ngu01a] and [Ngu02a] present a solution for Navier-Stokes equations to simulate the flames but is time consuming. [Hon07a] combine both a Navier-Stokes simulation with detonation shock dynamics to get wrinkled flames and cellular patterns but also takes a long time to render. A near realtime fire simulation and control system that uses artist derived flame profiles is described by Lamorlette and Foster in [Lam02a]. The Perlin noise function and an artist specified flame profile is used to generate procedural fire by [Ful07a].

The groundbreaking 1970's work by [Har73a] and [Har76a] first introduce the Lattice-Boltzmann Model (LBM) methods which have become the basis of many non Navier-Stokes fluid simulations. [Wei02a] use overlapping textures with turbulence details employing LBM for motion of the flames. [Zha03a] also employ LBM to simulate fire fronts around solid objects.

Some models, such as [Lam02a], discuss in much detail how to compute the lighting effects. For example [Lam02a] uses an emitting sphere at each flame segment to generate lighting. It is assumed that the lighting details are automatically handled at the renderer level which combine several global illumination algorithms like radiosity, ray tracing, and/or photon mapping. Importance sampling using volumetric illumination has been used by [Zha11a]. GPU simulation with volumetric data creates a variety of realistic and detailed fire simulation such as moving fire [Hor09a],

Radiance maps are images that store the intensities of light passing through each pixel. The direction of the light is determined by the projection used to create the image. They are used in high dynamic range imagery (HDRI) as described by [Deb97a]. Methods such as precomputed radiance transfer (PRT) first introduced by [Slo02a] use a spherical harmonics representation for low-frequency radiance computation. These spherical harmonics representations approximate a high resolution HDRI radiance map and are very effective for relighting objects in an environment. Primarily they have been used for static scenes but the technique is expanding for dynamic scenes as well. For example, [Kri05a] relight architectural models in real-time with moving lights by combining precomputed point light source clouds.

3 THE POINT LIGHT MODEL

The predominantly implemented illumination model for fire in realtime applications is the point light source model. It is used widely because there is hardware support for the algorithm and also because it is easy to compute the shading value since the light position is subtracted from the vertex or fragment position to get the $\vec{L}$ vector that is used by a Lambertian illumination model. In some implementations, the intensity is constant with a distance based falloff function. It can become more sophisticated by varying the intensity of the light or adding a random perturbation to the coordinates of the light source. The intensity and position are often varied using a smooth noise or interpolation scheme. The easiest way to think of this is a person holding a simple light bulb with a rapidly sliding dimmer switch and a jittery hand. In Figure 1, you can see the smooth uniform intensity that the point light model has. The problem is that point light sources are isotropic, but many sources of light are anisotropic. We will now present the LDRM model which attempts to model the anisotropic features that fire and other natural phenomena possess.

4 THE LDRM MODEL

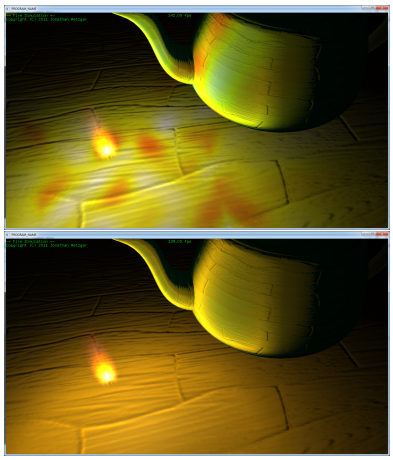

Figure 1: The LDRM method is compared to the point light source method. In the LDRM image, the intensity and frequency is turned up very high to clearly show the difference of the LDRM method and the point light source method. Realistic settings would be tuned to be more subtle.

Our Local Dynamic Radiance Map (LDRM) model stores a dynamically computed radiance map at coordinates P. The light emitted from P is cast in all directions simulating the first bounce of radiosity. The radiance may either be precomputed as an animation

or procedurally generated in realtime. A cube map or other similar abstraction is an ideal way of storing these radiance values.

We reproduce Kajiya's rendering equation [Kaj86a] below so we can illustrate how the LDRM fits into this standard model:

$$I(x,x') = g(x,x')[\varepsilon(x,x') + \oint \rho(x,x',x'')I(x',x'')dx'']. \quad (1)$$

The radiance of the LDRM is represented by $\varepsilon(x,x')$ while being directly affected by the visibility of the point x at point x' by the function $g(x,x')$. More specifically, the function $\varepsilon(x,x')$ is the radiance coming from direction $\vec{L}$ where $\vec{L}$ is the vector from the position of the light source to the point x, and is the direction of the sample. This value can be obtained by looking into the radiance map using the direction provided. For example, most graphics hardware have the ability to easily look up this value from a cube map using a vector as an input.

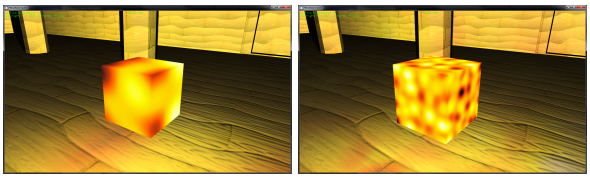

Figure 2: Two LDRMs generated using different frequencies of noise and their corresponding effect on the environment.

From Kajiya's rendering equation, we then map this to a simplified model where we can incorporate our rendering algorithms. Since we are not simulating the integral in his equation, we decided to make that a constant value and focus on just ε and g which is just the radiance of the light source and the visibility of the light source with the surface being illuminated. Essentially, the LDRM acts like a Gaussian surface in the sense, that instead of trying to compute the interactions with the actual flames or other phenomena and the surrounding environment, we perform an intermediate step of mapping it to a surface we can easily use in a rendering situation. This is clearly seen in Figure 3.

The incoming radiance which we will now call R, is then divided into the specular and diffuse reflection colors $k_{specular}R$ and $k_{diffuse}R$, respectively where $k_{specular} + k_{diffuse} = 1$. Depending on the reflectance model used, $k_{specular}$ and $k_{diffuse}$ may be computed differently. We decided to implement the Cook-Torrance model and we will discuss later in section 4.1 how to compute these values.

The dynamic radiance of a torch fire is approximated by using Ken Perlin's noise function. The radiance of each direction of the torch fire is computed and stored inside the radiance map. The unit vector l is used as input to the Perlin noise function and the resulting value is used to look up the color associated with the radiance.

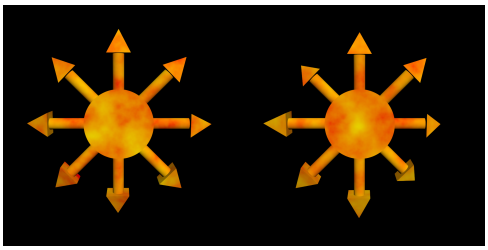

Figure 3: The LDRM method (right) differs from the point light source method (left) by modeling the light's outgoing radiance as a function of time, intensity, *and* direction.

A blackbody radiation color map is used to give color to the torch fire. An example color map is shown in Figure 7.

The LDRM is flexible. If a variety of natural phenomena used the same kind of simulation algorithm but differed only in color, a different color map will easily allow for adjusting that. For example, fire could probably use the same simulation code, but the specific chemical combustion properties would be approximated by mapping the resulting intensity values with the appropriate color map.

In our implementation, we have chosen Perlin noise because it generates smooth noise that can be animated and provides enough variety for our ideas to be implemented. Generation of a LDRM function that simulates the unique properties of fire (or other phenomena) is most definitely a topic for future study but is outside the scope of our research. Later we will discuss this possibility, but our experiments with Perlin noise showed significant enough improvement in scene realism versus the traditional point light source method that we discussed in the last section.

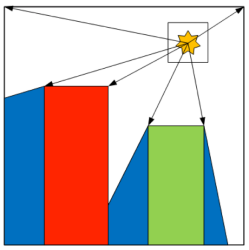

Figure 4: The LDRM projects radially from the center of the light position. Areas in blue get ambient lighting while others get direct illumination.

Figure 4 shows a diagram of how the LDRM method works. The box located around the position of the light represents the cube map. The arrows emitted from the center of the light through the box will look up the appropriate radiance and project it on the environment. If the area is in shadow (represented by shaded blue areas) then an ambient algorithm can determine the final illumination of those fragments. The containing rectangle

and the red and green rectangles represent the environment and objects visible to the light source. Figure 2 shows two example LDRMs generated using two different frequencies of noise. It can be easily seen how the LDRM works in a practical sense by observing that variation in intensity in the environment corresponds to the frequency of noise.

5 IMPLEMENTATION

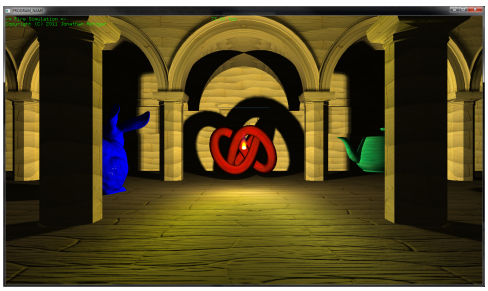

Figure 5: A rendering of the fire flicker effect program.

Our fire flicker effect simulation presented in this paper is designed to employ the LDRM model. A variety of rendering algorithms is used to simulate global illumination and a screenshot is shown in Figure 5. The global illumination algorithm implements the Cook-Torrance model, Blinn bump mapping, and cube map shadow mapping. It uses a simple ambient function that approximates indirect illumination by scaling the direct illumination by the amount of shadow present at that pixel location. The Cook-Torrance model allows for physically based illumination while the bump mapping algorithm allows for increased higher-frequency pseudo details. Finally instead of rendering the LDRM cube map in the scene, flames are dynamically computed and rendered into 2D textures and drawn onto rectangles in a fan like structure to give the torch a 3D look. In effect, the torch fire is used as an aesthetic place mat to show where the LDRM is located in the scene.

5.1 Illumination and Shadow Model

The Cook-Torrance model was chosen because it is a physically based model. Other models can easily be integrated as desired. The LDRM was used to supply the specular color $k_{specular}R$ for the Cook-Torrance model. This color is mixed in with the surface color of the fragment being rendered and scaled by the dot product of the surface normal and incoming light vector $\vec{L}$.

Since a torch is an omni-directional light source, cube mapped shadow mapping was selected to render the shadows. It is a fairly straightforward algorithm to implement, but it does take some tweaking to get the highest image quality. We chose to write a scalable multi-sampled shadow algorithm that we could adjust to measure performance of our technique. The number of samples can go from one sample to $N = 257$ samples. Here N can be varied based on the hardware capabilities of the system. The penumbra width is adjustable as a constant parameter in the shader program. Figure 6 shows two screenshots of the program using 1 sample shadows and 257 sample, wide penumbra shadows.

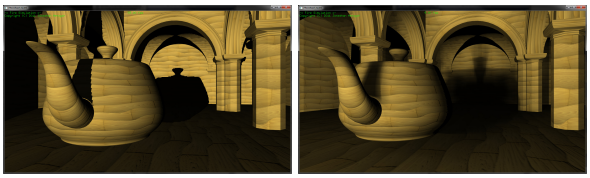

Figure 6: These two images shows a basic one sample shadow and a wide penumbra, 257 sample shadow.

5.2 Radiance Cube Map Generation

Perlin noise is a simple solution to generating non-linear radiance that is repeatable, smooth, and fluid. Depending on application performance, the noise can either be generated on the GPU or CPU. The fire program implemented in this paper used the GPU and a render-to-texture set up to render the six sides of a cube map. The GPU code for generating Perlin noise was implemented by [Gus06a] which we slightly modified to adjust for noise scaling and animation parameters used in the fire program.

The six textures are used as a cube map in the final rendering pass by the global illumination shader. The gray scale output of the radiance is then converted from heat values to RGB values by looking up the data in a color look up table. The texture is updated once per frame or as needed to maintain a target frame-rate. The color look up table is shown in Figure 7 and one side of a LDRM is shown in figure 8.

Figure 7: The color look up table mapping heat to their corresponding RGB values.

Figure 8: One face of a cube map generated by the LDRM method.

Special care needs to be taken to balance the noise so that it adds a subtle lighting effect to the scene. If the frequency of the noise is too high then the effect may look overdone where it can quickly be distracting. On the other hand, using hardly any noise or under using the effect will look as if the effect is not being used, so careful balancing needs to done to find a good range where the effect will be effective. Figure 2 shows this effect in

practice. Note how the high frequency map may make the environment look splotchy which is not realistic.

The LDRM may be used to create good lighting effects, but it is not complete without some motion of the shadows in the environment. Perlin noise is used once again to compute a time-varying offset which we add to the original position of the light. This new position is used to render the shadow maps and lighting. The final product then has dancing shadows which enhance realism.

5.3 Global Illumination in the Simulation

The equation

$$C_{out} = \max\left(R_{ambient}, R_{shadow}\right) \cdot \left(k_{diffuse} + R_{specular} \cdot k_{specular}\right). \quad (2)$$

is the basis for the global illumination algorithm for the fire program. The $R_{ambient}$ term specifies the ambient intensity of the pixel, the R_{shadow} represents the contribution of any direct lighting occuring at the pixel, the $k_{diffuse}$ term is the color of the surface at that pixel, the $k_{specular}$ term is the color of the specular reflection of the pixel, and the $R_{specular}$ term is the amount of reflected light at the pixel.

The terms are all computed from four different algorithms. The first algorithm is bump mapping which calculates the normal of the pixel. The second algorithm is the Cook-Torrance model which calculates the specular reflectance values $R_{specular}$ and $k_{specular}$ of the pixel. The third algorithm is the Cube Map Shadow algorithm which allows for omni-directional point light sources. Finally, the fourth algorithm is an intensity falloff model for the $R_{ambient}$ term to model indirect light. These have been covered in detail in previous works, but integrating them together will be briefly explained in light of the equation to compute C_{out}.

5.4 Ambient and Shadow Term

The ambient term is a simple approximation based on a inverse falloff law from the distance to the fire. The ambient term R_a is computed by the formula

$$R_{ambient} = \frac{1}{4|L|}. \quad (3)$$

This equation is a variation based on the inverse power law $I = \frac{P}{4\pi r^2}$ which gives us a brighter overall light intensity which is normally lost unless you do a full on radiosity simulation to get the intensity back through indirect reflections. This gives us some of that light which is normally "lost" in a local illumination model.

The shadow term R_{shadow} is computed with the following formula:

$$R_{shadow} = \min\left(\vec{N}_{vertex} \cdot \vec{L}, \vec{N}_{bump} \cdot \vec{L}\right) * \frac{1}{n} \sum_{j=0}^{n} s_j \quad (4)$$

where $\vec{N}_{vertex}$ is the interpolated vertex normal, $\vec{N}_{bump}$ is the per pixel normal derived from the normal map, $\vec{L}$ is the incoming direction of the light source, n is the number of samples being used for the shadows, and s_j is the boolean result of comparing the jth pixel depth value to the light depth buffer which is either 1 or 0. Taking the minimum of the dot products eliminates bump mapping on polygons not facing the light.

Together the ambient and shadow terms are used to determine the minimum illumination level of the fragment to be rendered. A simple maximum function is used to choose the ambient term or shadow term. If a fragment is completely shadowed, then the ambient term is used, otherwise the fragment is in penumbra and has some illumination.

5.5 Diffuse and Specular Term

The diffuse term $k_{diffuse}$ is generated from the surface color or texture of the object. The specular terms $R_{specular}$ and $k_{specular}$ are the coefficient of the reflected light and its color, respectively. This is where we can incorporate the LDRM model. The $k_{specular}$ value is obtained by using the $\vec{L}$ vector as the lookup in the LDRM cube map. The $R_{specular}$ term is based off the Cook-Torrance model equation [Coo81a]

$$R_{specular} = \frac{F}{\pi} \frac{DG}{(N \cdot L)(N \cdot V)}. \quad (5)$$

F is the Fresnel term, D is the micro-facet distribution factor, G is the geometric attenuation factor, V is the view vector, and L is the vector from the light to the fragment. Additional details about using the Cook-Torrance may be found by referring to the original paper. It is important to note that the LDRM model is not just limited to Cook-Torrance, but may be incorporated with any illumination model.

5.6 CPU and GPU 2D Flame Simulation

Our flame rendering system is based off a simple cellular automata model to generate fire. This cellular automata is a simplified model of Lattice Boltzmann Methods (LBM) which originated with the work of [Har73a]. [Che98a]'s work summarize the developments of the model into its more current form. The method works by using a lattice structure representing the fluid to be simulated. A *convection operator and* collision operator transform the lattice over time and cause the fluid process to occur. The nineties demo scene fire effect used a simple averaging function to cause convection and simulate collisions. Figure 9 shows a screenshot that simulates this full screen fire effect.

This fire effect can be modified to generate small flames or torch sources. Further improvements can be made to increase precision as well. Typically this effect uses 8-bit integer mathematics to store the heat values. While

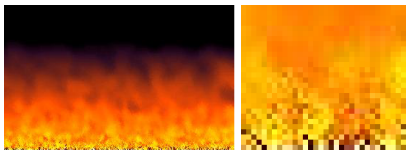

Figure 9: This image shows a fire simulation where the entire bottom row is used as the heat source and that use of integer math causes noisy artifcats near these randomized heat sources.

this is accurate enough for most of the effect, it results in artifacts near the source of the fire as shown in Figure 9. Changing the representation from integers to floating point math eliminates these artifacts and increases dynamic range. This can be coupled with established HDR techniques and physically based color computations for different chemical reactions.

The flame is generated by adding or seeding heat to points on the lattice. The flame will flow during the convection operator step. During the collision operator step, the flame mixes together. The three steps will cause the flame to take shape as this process repeats. A simple circular falloff model is used for seeding the heat to the fire. Notice in Figure 11 two different kinds of falloff patterns: the simple radial falloff used in the fire program and a noisy radial falloff used for the fires in Figure 10. By quickly changing the location of the falloff pattern, turbulence is created. The fire effect and varying levels of turbulence are shown in Figure 10.

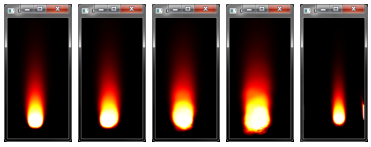

Figure 10: The radius of the circle in which the center of the flame source is moved causes a more turbulent flame. On the far right, improperly handled edges cause “heat sink” artifacts.

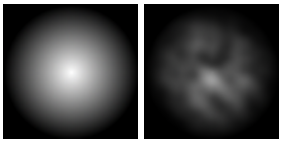

Figure 11: Two falloff patterns for seeding fires. The second pattern adds some turbulence to the resulting flames.

This fire effect can be computed using a GPU and a graphics based shader language (i.e. GLSL) was adequate for our simulation. The algorithm is shown in Listing 1 and is fairly straightforward. It determines whether the fragment is in the simulation area or not (which if not handled correctly creates “heat sinks" shown on the far right in Figure 10). Simulating diffusion and cooling is obtained by averaging several neighbor samples at each fragment and multiplying by a factor $life$, respectively. Heat is added to all the fragments located inside a circle (a “heat sink") which is randomly jittered according to the desired turbulence of the flame.

```
@VERTEXSHADER
uniform mat4 ProjectionMatrix;
varying vec2 uv;

void main() {\\
uv = gl_MultiTexCoord0.st;
    gl_Position = ftransform();
}

@FRAGMENTSHADER
#version 140
uniform sampler2DRect FireLattice;
uniform sampler1D radianceCLUT;
uniform float a, b, radius;
uniform float width, height;
uniform float heat, life;
uniform float turbulence;
in vec2 uv;
out vec4 gl_FragColor;
float rand(vec2 co) {
  return fract(sin(dot(co.xy ,vec2(
    12.9898,78.233))) * 43758.5453);
}
void main() {
  float x = uv.s, y = uv.t;
  float data = 0;
  float r2 = radius * radius;

  if (x >= 1 && x < width-2 &&
      y >= 3 && y < height-1) {
    if (x >= a-radius && x < a+radius &&
        y >= b-radius && y < b+radius) {
      float f = (x-a)*(x-a) + (y-b)*(y-b);
      if (f < r2) {
        data = texture(FireLattice,
                       vec2(x, y)).a;
        data += heat * (1 - f/r2);
     }
    }
    data+=texture(FireLattice,
                  vec2(x, y+1)).a;
    data+=texture(FireLattice,
                  vec2(x-1, y-1)).a;
    data+=texture(FireLattice,
                  vec2(x+1, y-1)).a;
    data+=texture(FireLattice,
                  vec2(x, y-2)).a;
    data = clamp(data * life / 4.0,
                 0.0, 1.0);
  } else {
    data = 0;
  }
  vec3 color2 = texture(radianceCLUT,
                        data).rgb;
  gl_FragColor = vec4(color2,data);
}
```

Listing 1: A GLSL Shader that computes the flame simulation.

6 RESULTS

The LDRM model takes up relatively little extra load in conjunction with normal rendering depending on the number of lights being used. The majority of performance loss comes from shadow mapping when large numbers of samples are being used. Rendering high resolution LDRM cube maps may also reduce performance but this can be mitigated by using low resolution maps when large numbers of lights are being used.

The benchmarks were conducted using a Windows 7 OS, Intel i7 930 2.80GHz processor with 6GB of RAM, and a NVIDIA GeForce GTX 480 graphics card with 1.5 GB of GDDR5 memory. Each benchmark was measured by recording the number of frames per second (FPS) once per second over a period of 25 seconds. The mean frame rate was then computed to filter noise in the readings, though the noise present was so low that it had an insignificant effect on the final numbers. Finally, we kept the frame rate as high as possible so we could ensure that our simulation would run on less capable graphics cards.

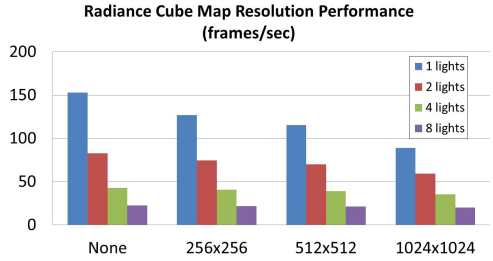

Figure 12: A comparison of radiance cube map size versus performance.

We tested the performance of our method using different resolution LDRM cube maps and by not rendering them at all. Figure 12 shows the results when 1, 2, 4, or 8 lights are being used. When using a 256x256 LDRM cube map, performance drops by 41%, 68%, and 83% for 2, 4, or 8 lights, respectively. When using 512x512 cube maps, performance drops by 39%, 66%, and 82% for 2, 4, or 8 lights, respectively. For 1024x1024 cube maps, performance drops by 33%, 60%, and 77% for 2, 4, or 8 lights, respectively. Compared to not using LDRMs at all, performance drops by 4% to 17% for 256x256 cube maps, 6% to 25% for 512x512 cube maps, and 11% to 42% for 1024x1024 cube maps.

Next, we tested the performance of our flame rendering system on the CPU and the GPU. Overall, the GPU had a clear lead in performance especially as resolution is increased. However, until much higher resolutions of flame simulations are used, the number of flames rendered per second on the CPU was in the hundreds which is sufficient. It is also clearly shown that using the GPU in conjunction with the CPU yielded little decrease in overall performance. We are unable to easily compare

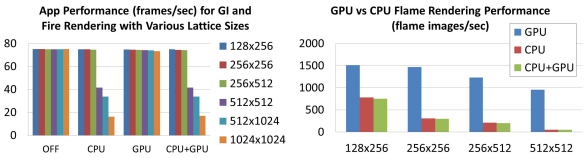

Figure 13: On the left, we see that application performance is not affected until high resolution lattice sizes are used. On the right, the GPU far surpasses the CPU in raw fire rendering performance.

our flame rendering algorithm with others because ours is not volumetric and has very strong boundary conditions which make it incapable of handling interactions in a 3D environment which a volumetric simulation could.

When integrated with the global illumination simulation, the GPU advantage becomes more obvious. CPU performance drops off fast when using high resolution lattice simulations, though it is almost unnoticeable when using reasonably sized maps. In contrast, the GPU simulations have a very small performance penalty when using large lattices. It should be noted that multithreading was not used in the CPU simulation, but the GPU still has enough compute power for the large lattice sizes that even an 8 core CPU could not outperform it. Figure 13 shows the performance graphs for running the global illumination simulation and rendering the flames with either the CPU, GPU, or both. It also shows the baseline performance of the GI simulation without rendering the flames. Effectively, you get the flame rendering for free for small resolution flame images.

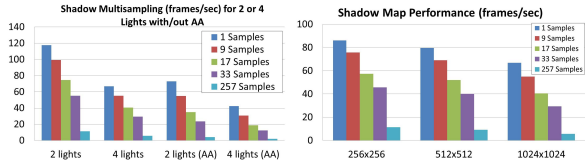

Figure 14: On the left, antialiasing halves overall performance. On the right, reasonable numbers of shadow samples still allow interactive frame rates.

Finally, we examine the performance of the global illumination algorithm. Figure 14 shows the performance of the global illumination algorithm both when using multi-sampled shadows and different resolution shadow maps, respectively. When using a reasonable number of lights, it is very easy to obtain very interactive rates. Performance drops quite a bit when using anti-aliasing, but the image quality is greatly improved and small pixel artifacts that show up when not using anti-aliasing almost entirely disappear.

Shadow quality is very good at 33 samples per pixel and the frame-rate is quite interactive. Wide penumbras are allowed which increases the realism of far off shadows where the area lighting effect of the flames would not

create sharp edges. The shadow map size affects performance but not as dramatic as varying the number of samples. Memory usage does increases quickly so tweaking is necessary to determine the lowest acceptable shadow map resolution.

Adding motion to the shadows does a good job of distracting the observer from noticing some minor problems with shadow mapping. Some of these problems include light leakage or surface acne. Ultimately, the moving shadows create the realistic appearance that the fire has on the scene while the LDRM model adds a subtle ambience to the scene, that when switched off, makes the simple intensity modulation based fire flicker effect seem somewhat lifeless.

7 CONCLUSION AND FUTURE WORK

The LDRM model presented in this paper helps add realism to scenes where torch fires are being used. The ambiance created by using shifting anisotropic illumination patterns add subtle depth and realism to scenes compared to the simple point light source model. The performance penalty is small and the algorithm is trivial to implement for any realtime graphics engine.

Future study of LDRMs to enhance direct illumination is promising and is an excellent extension to normal precomputed radiance transfer. In the future we are looking into simulating a volumetric fire and comparing the actual radiance with our approximation. We believe that creating LDRM models of other nonlinear light sources would be highly beneficial towards accurately simulating other phenomena in a realtime application. Finally, we are looking into using spherical harmonics as a substitute for cube maps which may allow LDRMs to be used in resource limited environments.

8 REFERENCES

[Che98a] Chen, S., and Doolen, G.D., *Lattice boltzmann method for fluid flows.* Annual Review Fluid Mechanics, 1998, pp.329-364.

[Coo81a] Cook, R. L., and Torrance, K. E., *A reflectance model for computer graphics.* Proceedings of the 8th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '81, 1981, pp. 307–316.

[Deb97a] Debevec, P. E., and Malik, J., *Recovering high dynamic range radiance maps from photographs.* Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '97, 1997, pp. 369–378.

[Ful07a] Fuller, A. R., Krishnan, H., Mahrous, K., Hamann, B., and Joy, K. I., *Real-time procedural volumetric fire.* In Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM, New York, NY, USA, I3D '07, 2007, pp. 175–180.

[Gus06a] Gustafson, S., *Dsonoises, a set of useful functions for sl.*, 2007. url:http://staffwww.itn.liu.se/~stegu /aqsis/DSOs/DSOnoises.html

[Har73a] Hardy, J., Pomeau, Y., and de Pazzis, O., *Time evolution of a two-dimensional classical lattice system.* Phys. Rev. Lett. 31, 5, 1973, pp. 276–279.

[Har76a] Hardy, J., de Pazzis, O., and Pomeau, Y., *Molecular dynamics of a classical lattice gas: transport properties and time correlation functions.* Phys. Rev. A 13, 5 (May), 1976, pp.1949-1961.

[Hon07a] Hong, J.-M., Shinar, T., and Fedkiw, R., *Wrinkled flames and cellular patterns.* In ACM SIGGRAPH 2007 papers, 2007, ACM, New York, NY, USA, SIGGRAPH '07.

[Hor09a] Horvath C and Geiger W., *Directable high Resolution simulation of fire on the gpu.* In ACM SIGGRAPH 2009 papers, 2009, ACM, New York, NY, USA, SIGGRAPH '09, 28(3).

[Kaj86a] Kajiya, J. T., *The rendering equation.* In ACM SIGGRAPH 1986 papers, ACM, New York, NY, USA, SIGGRAPH '86, 1986, pp. 143-150.

[Kri05a] A. W., Akenine-Möller, T., and Jensen, H. W., *Precomputed local radiance transfer for real-time lighting design.* ACM Trans. Graph. 24, July 2005, pp. 1208-1215.

[Lam02a] Lamorlette, A., and Foster, N. *Structural modeling of flames for a production environment.* Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '02, 2002, pp. 729-735.

[Ngu01a] Nguyen, D. Q., Fedkiw, R. P., and Kang, M. *A boundary condition capturing method for incompressible flame discontinuities.* Journal of Computational Physics 172, September, 2001, pp. 71–98.

[Ngu02a] Nguyen, D. Q., Fedkiw, R., and Jensen, H. W. *Physically based modeling and animation of fire.* Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '02, 2002, pp. 721–728.

[Slo02a] Sloan, P.-P., Kautz, J., and Snyder, J. *Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments.* Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '02, 2002, pp. 527–536.

[Wei02a] Wei, X., Li, W., Mueller, K., and Kaufman, A. *Simulating fire with texture splats.* Proceedings of the conference on Visualization '02, IEEE Computer Society, Washington, DC, USA, VIS '02, 2002, pp. 227–235.

[Zha03a] Zhao, Y., Wei, X., Fan, Z., Kaufman, A., and Qin, H. *Voxels on fire.* Proceedings of the 14th IEEE Visualization 2003 (VIS'03), IEEE Computer Society, Washington, DC, USA, VIS '03, 2003, pp. 36.

[Zha11a] Zhang, Y., Zhu, D., Qiu, X., Wang, Z. *Important Sampling for volumetric illumination of flames.* Visual Computing in Biology and Medicine, VR in Brazil, Computer & Graphics, 35(2), 2011, pp. 312-319.