GPU TECHNOLOGY CONFERENCE

Flame On: Real-Time Fire Simulation for Video Games

Simon Green, NVIDIA
Christopher Horvath, Pixar

Introduction

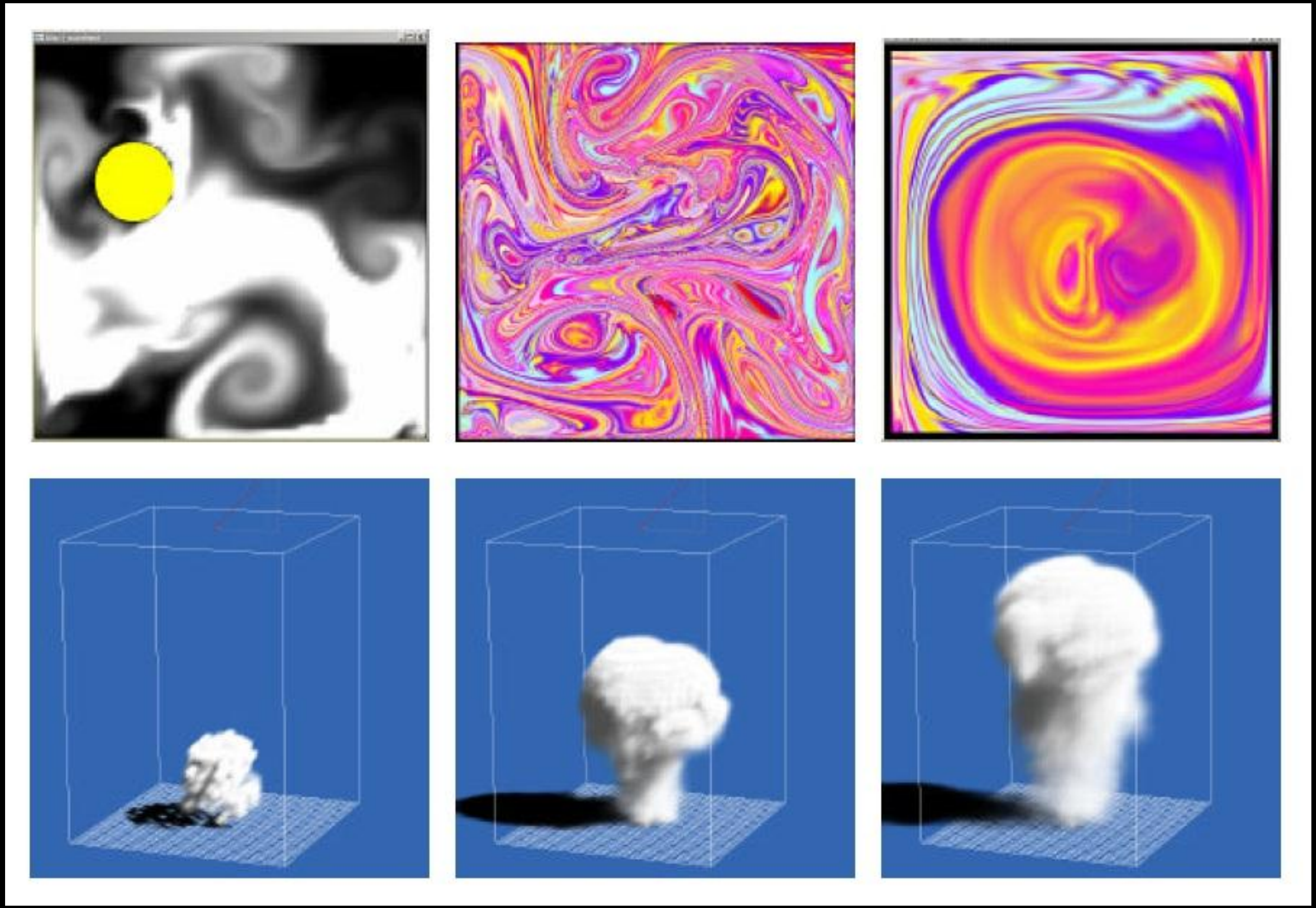
- This talk is about achieving realistic *looking* simulations for games / visual effects
  - Not necessarily physically accurate!
- There is a large artistic component

Overview

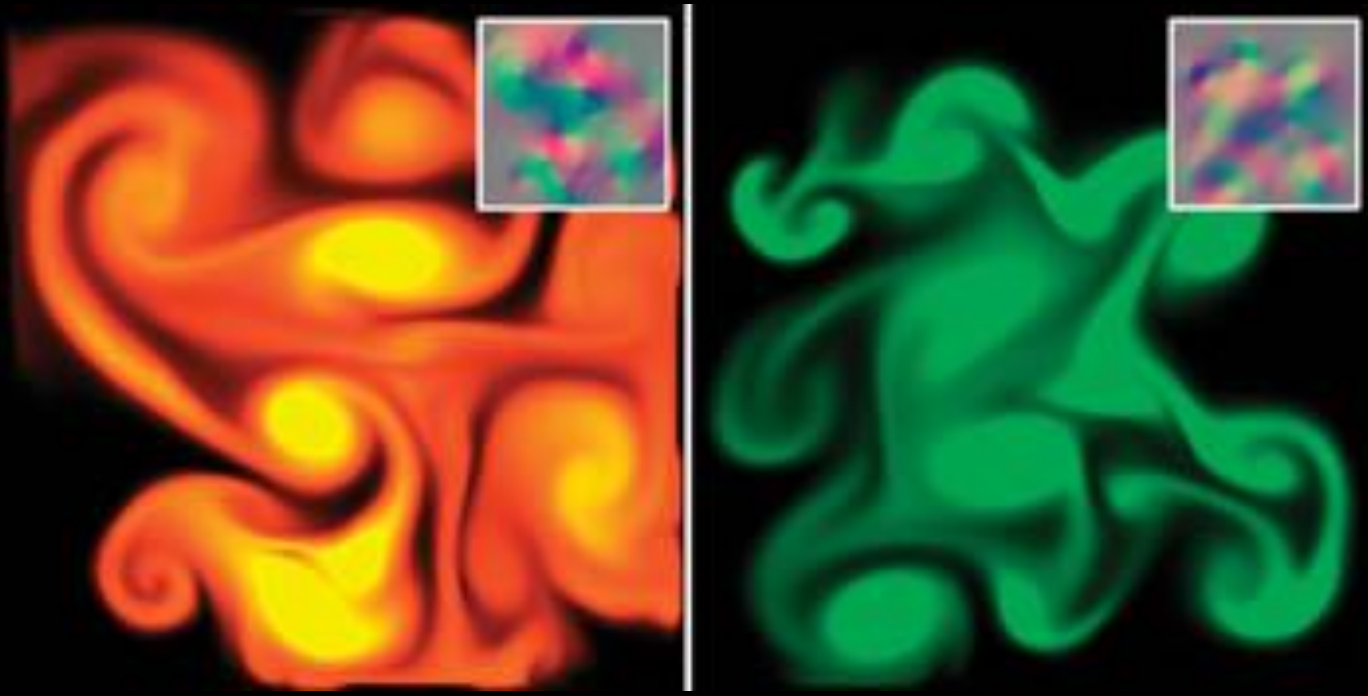
- 2D fire simulation using CUDA
- Sneak peak: 3D fire simulation using DirectX 11
- 5 Tips For Good Looking Fluid Sims

A Brief History of Eulerian Fluids on the GPU

“Stable Fluids”, Jos Stam, Siggraph 1999



Mark Harris' 2D fluid solver (GPU Gems 1, 2004)



3D fluid solver (GPU Gems 3, Crane, Llamas, Tariq, 2007)

APEX Turbulence
(Cohen, Tariq, 2010)

Interactive Fluid-Particle Simulation using Translating Eulerian Grids

Inspiration

- “Directable, high-resolution simulation of fire on the GPU”, Horvath, Geiger, SIGGRAPH 2009
- Computes high-res 2D slices of a 3D simulation
- Seeded using particle system
- GPGPU - used OpenGL
- Used in Harry Potter film

Goal - Interactive Fire for Video Games

- Most video games today use 2D sprites for fire
  - Procedural, or based on filmed footage
- 3D simulation probably still too expensive for real-time use today?

Today’s Video Game Fire

Simulated Fire

- Advantages
  - High resolution
  - Non-repeating animation
  - Can respond to wind etc.
  - Less storage (?)
- Disadvantages
  - Computation time
  - Artist controllability

Implementation

- Implemented 2D stable fluids solver in CUDA
  - Uses pitch-linear textures to store fields
  - `cudaMallocPitch / cudaBindTexture2D`
- Geometric multi-grid solver
  - Credit: Nuttapong Chentanez
- OpenGL for rendering
  - Shading done in GLSL pixel shader

Example CUDA Kernel

```
__global__
void pressureSolveD(float * __restrict__ newPressure,
                    const float * __restrict__ divergence,
                    int width, int height,
                    int pitch)
{
    int x = blockIdx.x*blockDim.x + threadIdx.x;
    int y = blockIdx.y*blockDim.y + threadIdx.y;
    int i = y*pitch+x;
    if (x >= width || y >= height) return;

    float2 pos = make_float2((float)x + 0.5f, (float)y + 0.5f);
    float pL = tex2D(pressureTex, pos.x - 1, pos.y);
    float pR = tex2D(pressureTex, pos.x + 1, pos.y);
    float pB = tex2D(pressureTex, pos.x, pos.y - 1);
    float pT = tex2D(pressureTex, pos.x, pos.y + 1);

    float bC = divergence[i];
    float pNew = (pL + pR + pB + pT - params.dx2*bC) * 0.25f;

    newPressure[i] = pNew;
}
```

Fire Recipe

- Take smoke simulator
  - Velocity, density
- Add new channels
  - Temperature, Fuel, Noise
- Add a simple combustion model
  - Combustion consumes fuel, generates heat
  - Heat also generates upwards buoyancy force

Tip 1 - Get the Colors Right

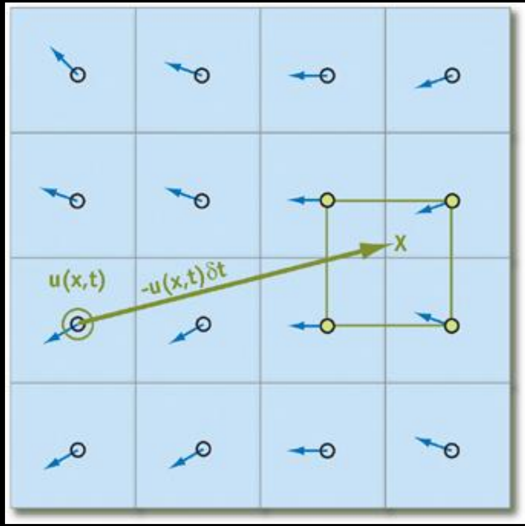
- Need to map temperature to color
  - use physically-based black body radiation model (see later)
  - Or: just an artist defined color gradient
- Dynamic range is important
  - Fire is very bright!
- Can apply curve to density to get sharp flame edges

Temperature

Color

Tip 2 – Use High Quality Advection

- Advection determines quality of motion and appearance
  - detail in velocity and density fields
- Bilinear filtering not really good enough
  - To much blurring over time
- Lots of other options:
  - Higher-order filters (cubic)
  - Error correction schemes – e.g. MacCormack
  - Particle based - PIC/FLIP
- We used Catmull-Rom filter, bounded to neighbourhood



GPU TECHNOLOGY CONFERENCE

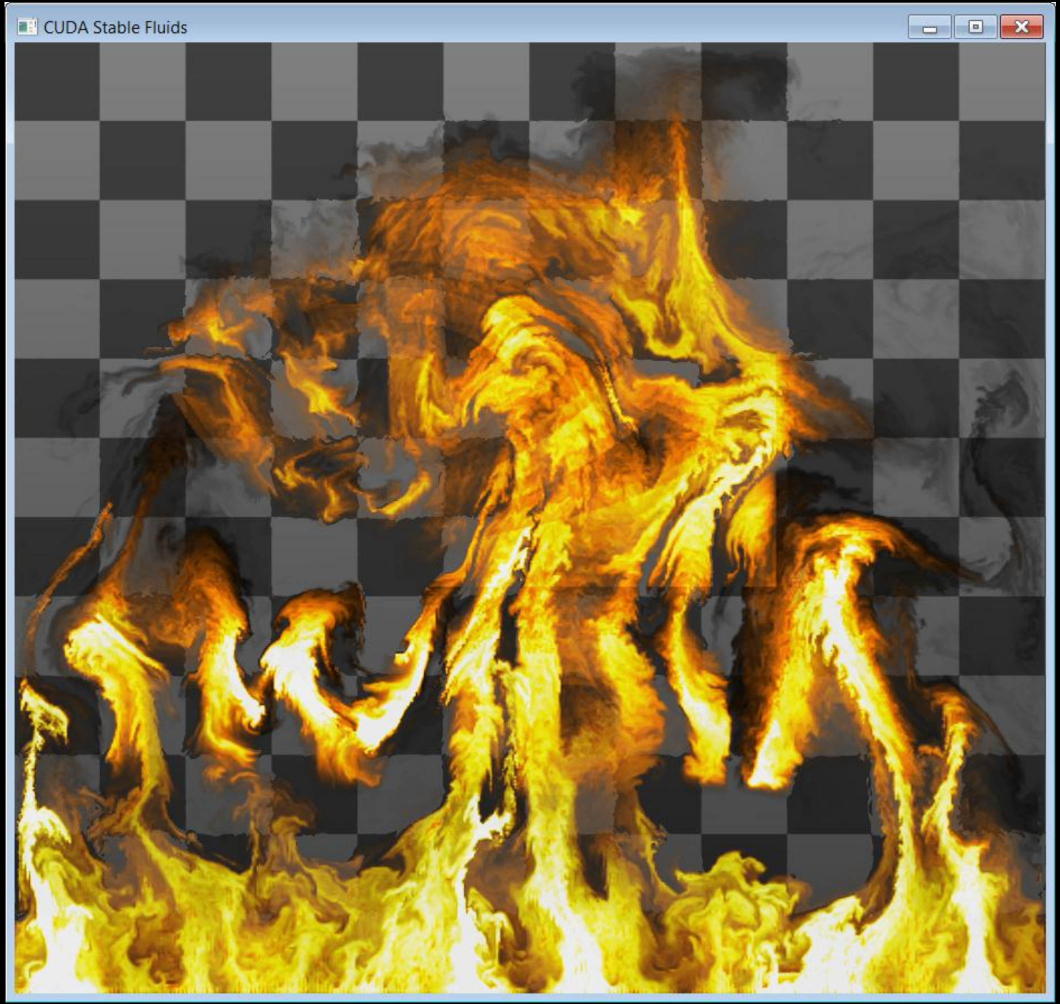
Tip 3 – Use a High-Res Density Field

- Density field can be much higher resolution than velocity field
  - 4x or more
- Read interpolated velocity field when advecting density
- Need to downsample density to velocity resolution if simulation is coupled
  - i.e. buoyancy based on density

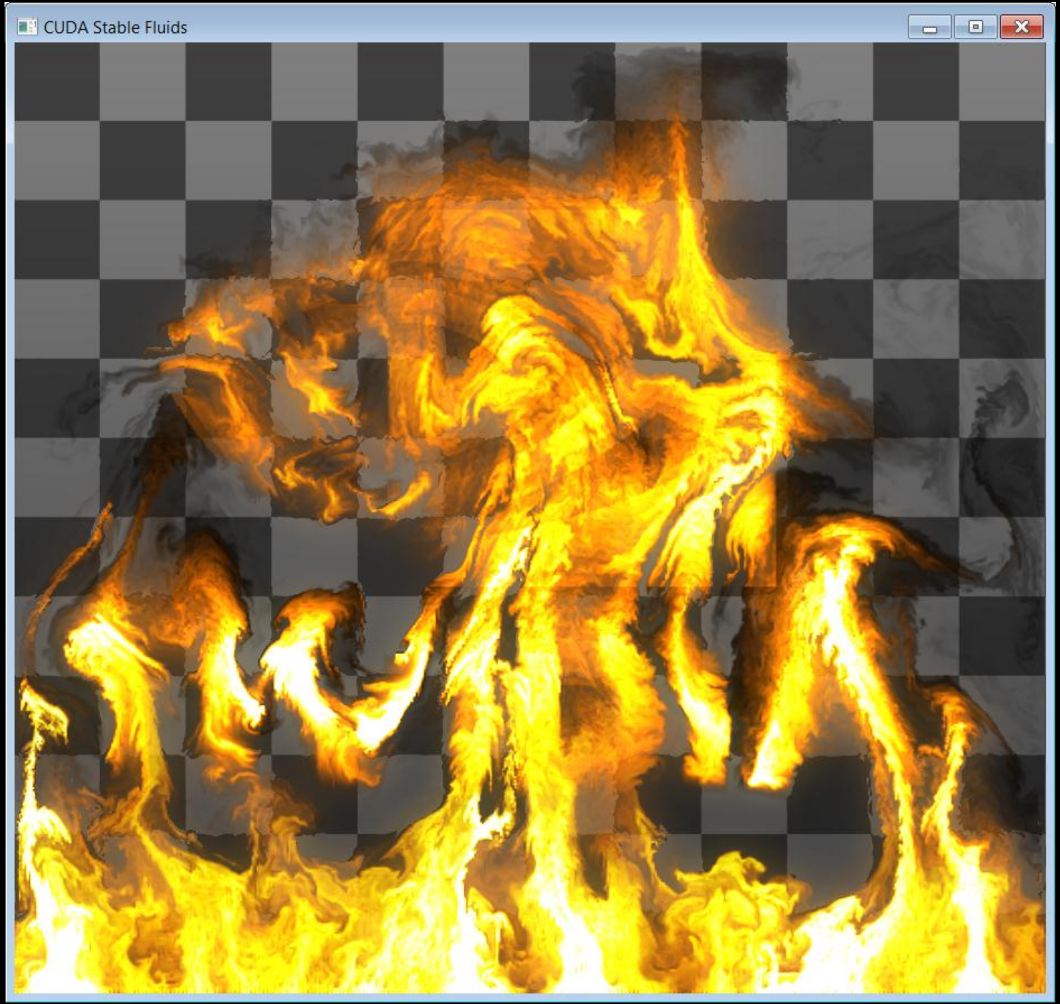
Tip 3 – Post Processing is Important

- Fire is very hot!
- Use post-processing to communicate temperature to viewer
  - Glow - blur HDR image, add back on top
  - Heat distortion – offset background based on gradient of temperature
- Motion blur
  - sample image several times along velocity vector

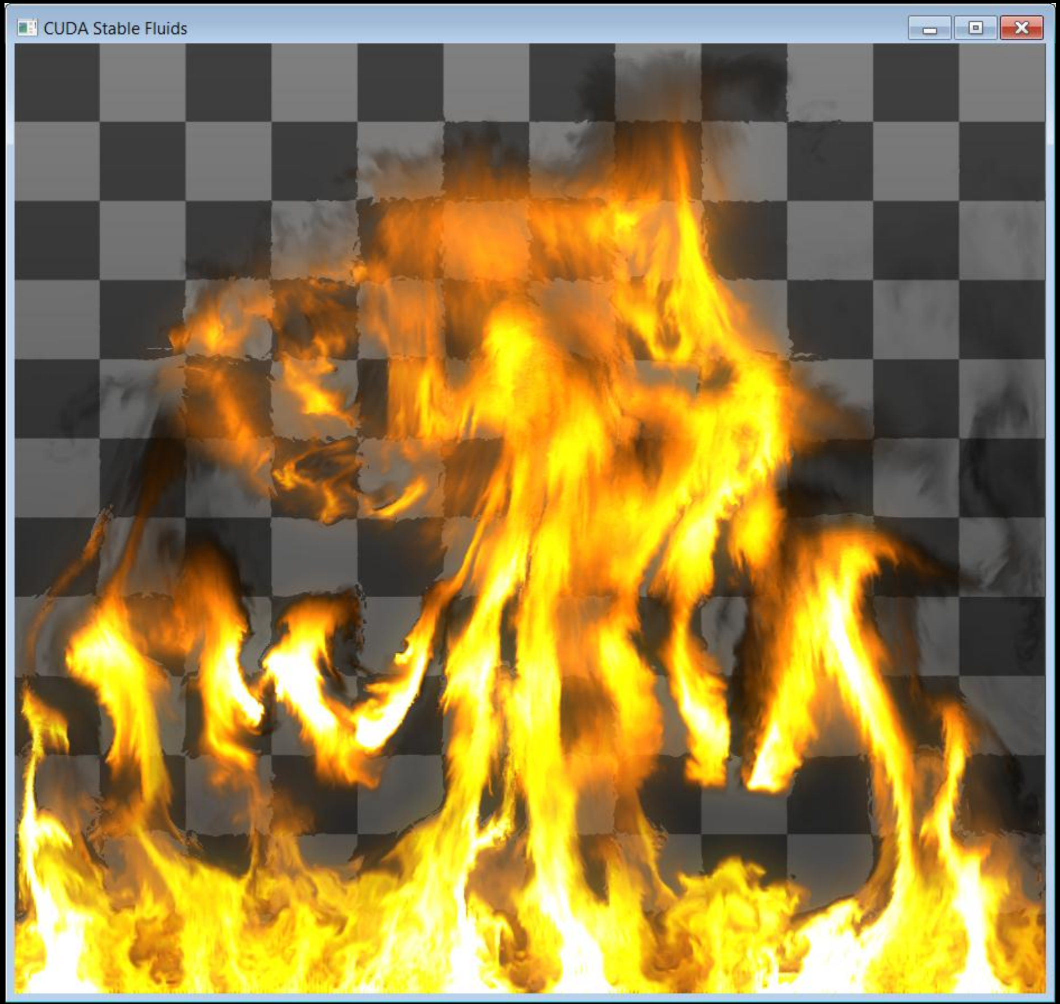
No Glow



With Glow

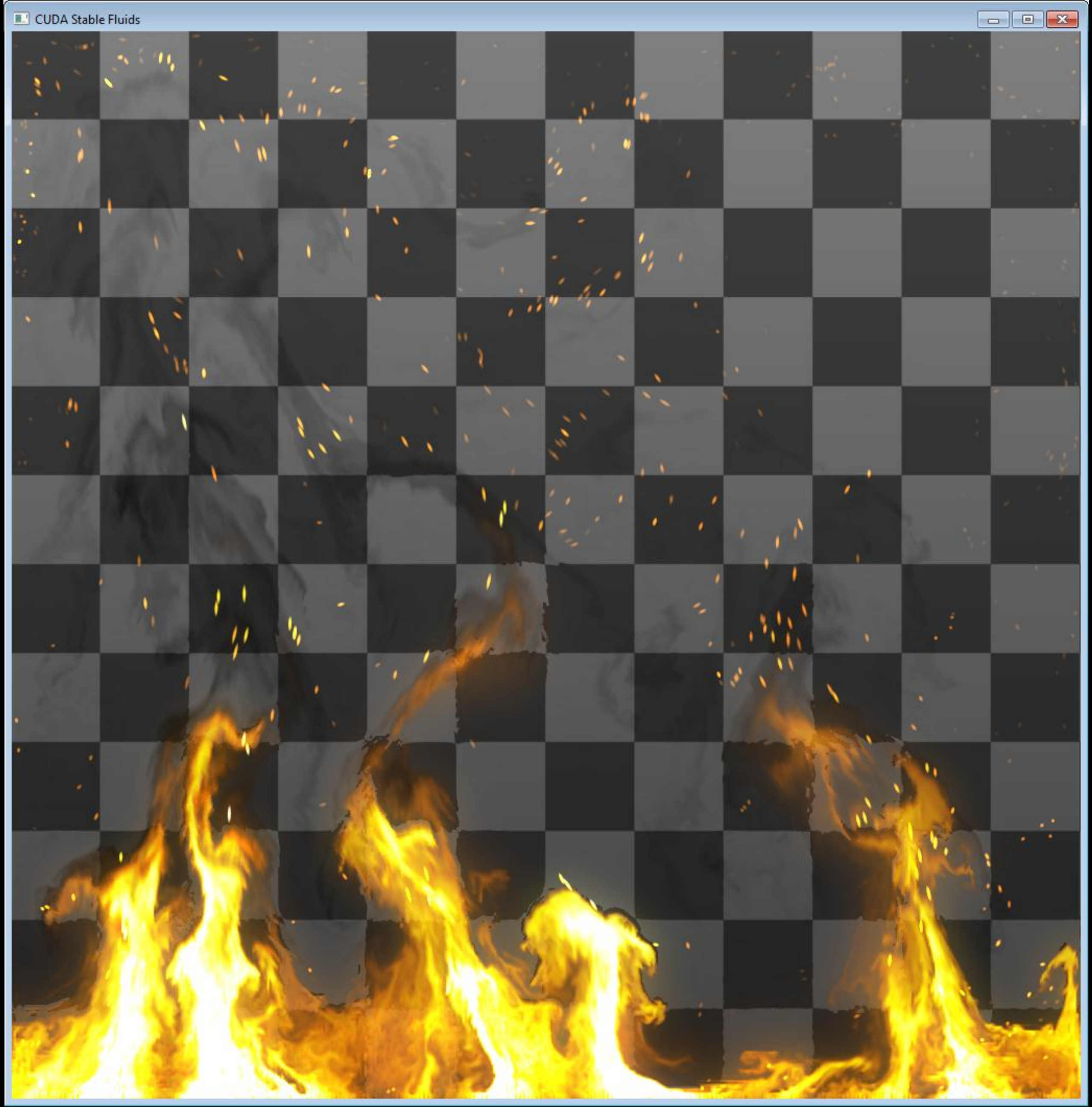


With Motion Blur



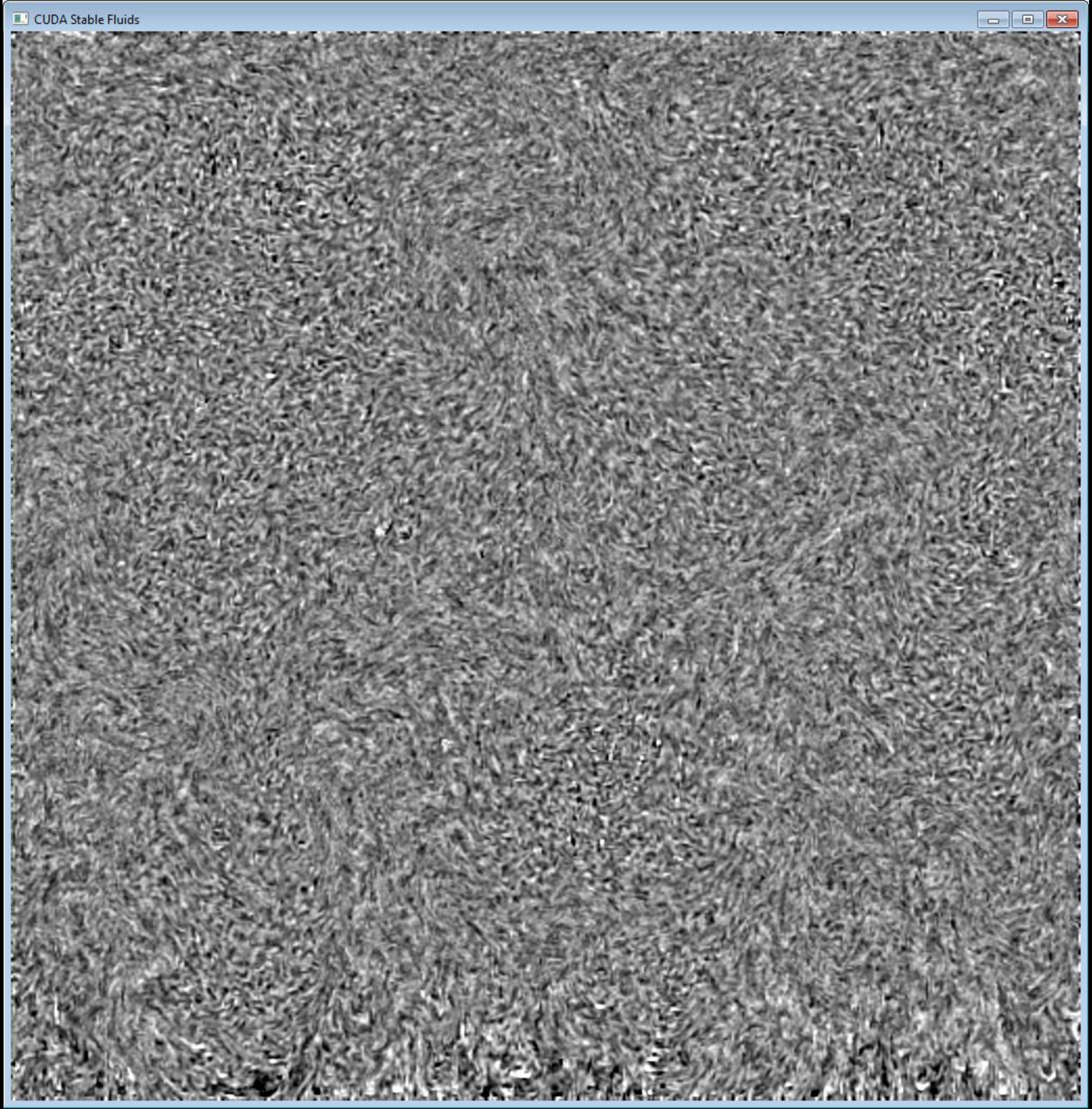
Tip 5 - Embers

- Add particles passively advected by velocity field
- Shows motion of air even in empty regions
- Motion blurred
  - Drawn as quads stretched between previous and current position (using geometry shader)
- Inherit temperature from simulation
  - Cool over time


CUDA Stable Fluids

Tip 4 - Just Add Noise

- Fire is very turbulent and fast moving
- Use high levels of vorticity confinement to preserve vortices
- Use procedural (curl) noise to add turbulence
- Also advect a 2D noise field
  - Blend in small amount of noise each frame
  - Can be used to add detail to other fields
  - Noise moves with fire


CUDA Stable Fluids


CUDA Stable Fluids

Work in Progress - 3D Simulation

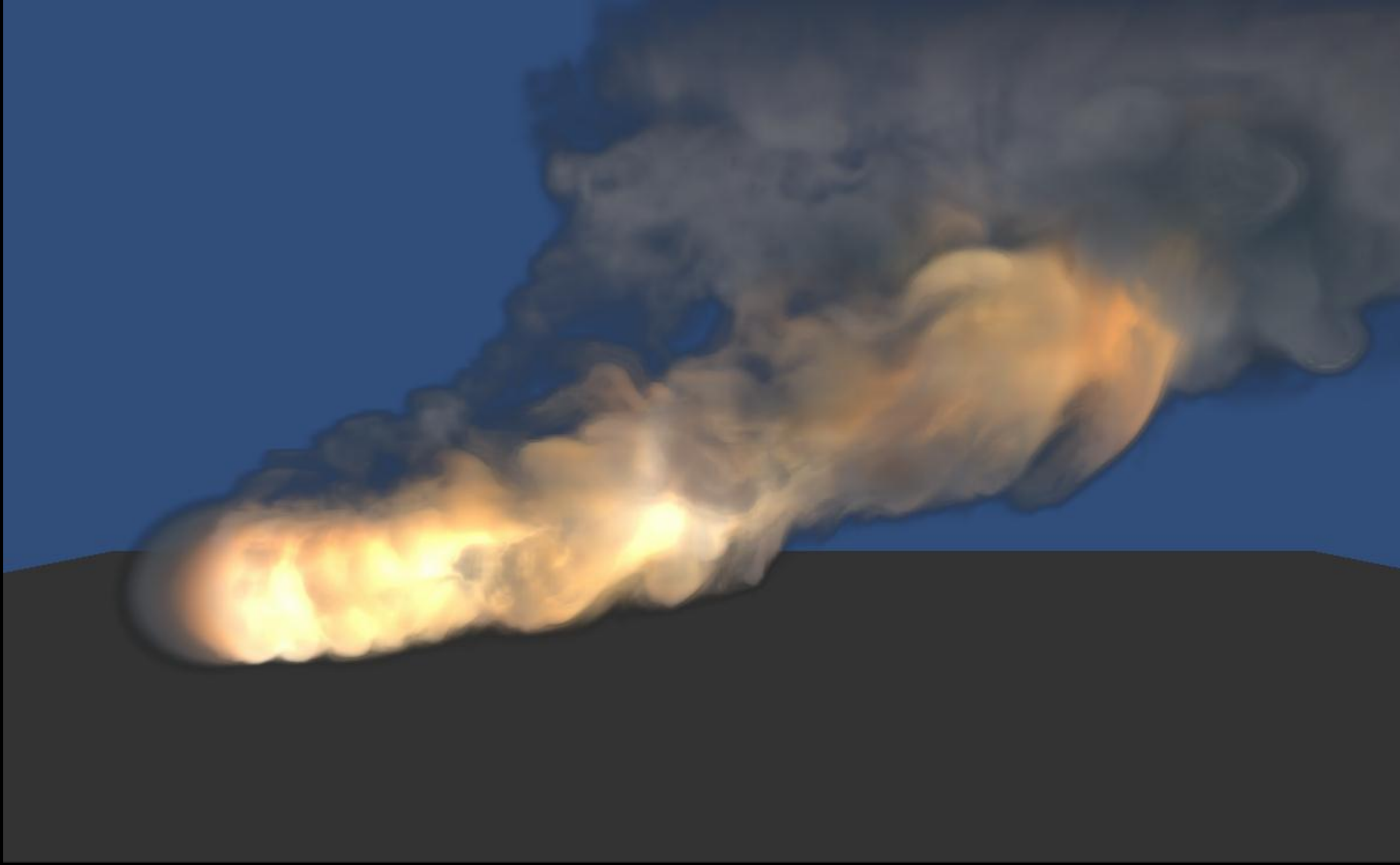
- Relatively simple to extend simulation to 3D
- Surface writes to 3D textures are now possible
  - in CUDA 4.x and DirectX 11

3D Performance

- Texture performance is great on Kepler architecture
- Sample results:
  - 128 x 128 x 64 (0.5M) voxels for sim, 64 solver steps
  - 2x res density field (8M voxels)
  - 17 msecs per frame, including rendering
  - (GeForce GTX 680)

Tip 5 – Add Light Scattering

- Simple scattering approximation
  - Similar idea to Light propagation volumes
  - (Discrete Ordinate Method)
- Basic algorithm:
  - Render radiance to 3D texture
  - Blur radiance in 3 dimensions
  - Sample blurred radiance (indirect light) in volume render



Demo

Physically Correct Flame Rendering

Or… "How to get the Planck Blackbody Radiation Function to actually look right."

Overview

- Components of Flame Appearance
  - Blackbody Radiation
  - Spectral Emission
- Tristimulus Response
  - CIE XYZ
  - Direct RGB (Human,Camera,Infrared)

Flame Appearance

- Blackbody Radiation of Combustion Byproducts (Soot/Smoke) - The Red/Orange/Yellow part.
- Spectral Emission - The Blue/Purple/Green part.
- http://en.wikipedia.org/wiki/Flame

Blackbody Radiation

Planck's Law

$$B_\lambda(T) = \frac{2hc^2}{\lambda^5} \frac{1}{e^{\frac{hc}{\lambda k_B T}} - 1}$$

- Relationship between emitted radiant intensity at each individual wavelength of light with temperature
- Calculated for a spectrum of wavelengths within visible range (380nm to 780nm). With 5nm increments, this is a color sample with 81 values.

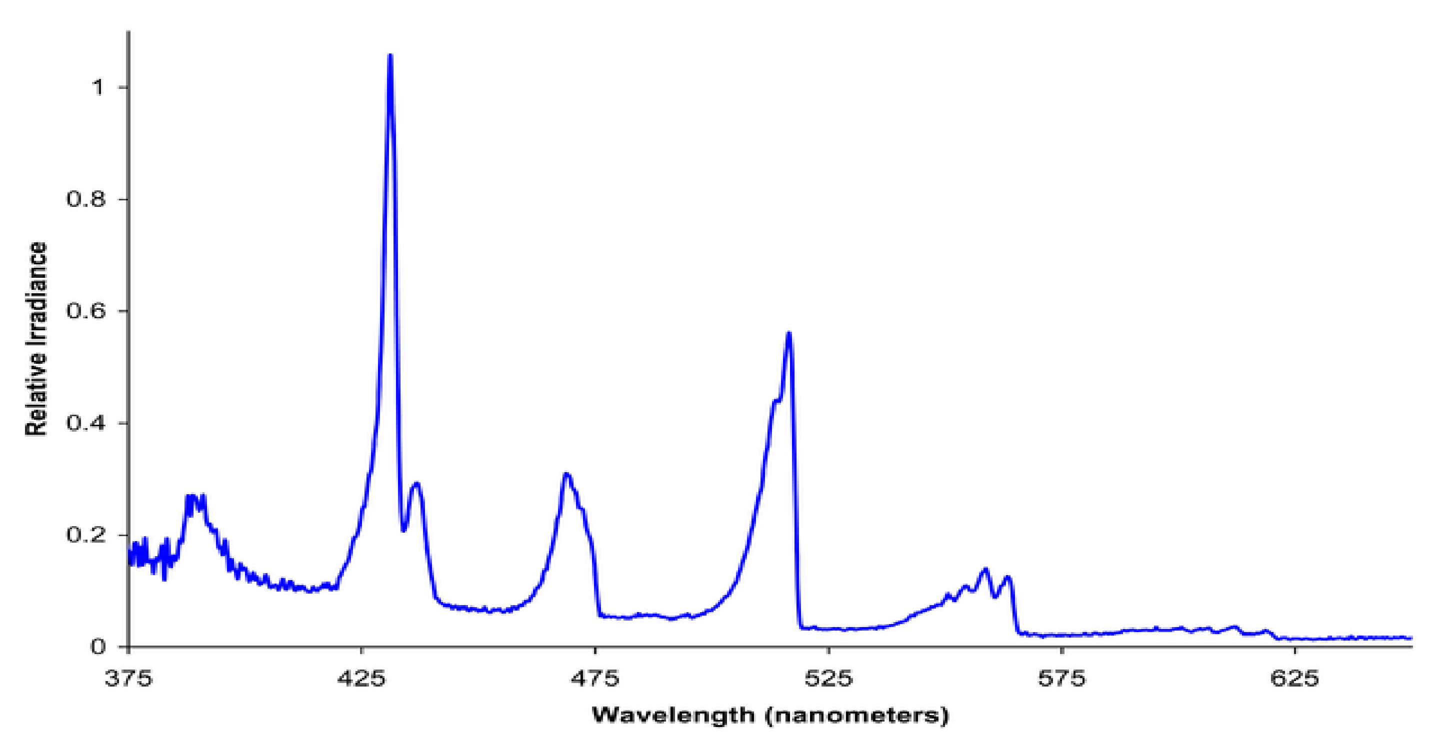
GPU TECHNOLOGY CONFERENCE

```
//-*************************************************************************
// The Gas Constant (R) is the amount of energy per unit temperature increment
//  (Kelvin) per mole, and has units Joules per mole Kelvin
//
// Avagadro's number is the number of elementary particles per mole,
// and has units mol^-1
void BlackbodySpectrum::setTemperature( double i_temperature )
{
    m_temperature = i_temperature;

    // This is the Planck Blackbody Radiation Function.

    static const double speed_of_light = 2.99792458e8; // m/s
    static const double plancks_constant = 6.6260755e-34; // J*s
    static const double gas_constant = 8.314462175; // J/(mol K)
    static const double avagadros_number = 6.0221412927e23; // 1/mol

    // boltzmanns' constant is the gas constant divided by
    // avagadros number.
    // It is the amount of energy per unit temperature increment,
    // per elementary particle.
    static const double boltzmanns_constant =
        gas_constant / avagadros_number; // J/K

    static const double constant_1 =
        plancks_constant * speed_of_light * speed_of_light / 2.0;
    static const double constant_2 =
        plancks_constant * speed_of_light / boltzmanns_constant;

    for ( int i = 0; i < Spectrum::N; ++i )
    {
        double lambda = Spectrum::wavelength( i );
        m_spectrum[ i ] = constant_1 /
            ( lambda * lambda * lambda * lambda * lambda *
              ( exp( constant_2 / ( lambda * i_temperature ) ) - 1.0 ) );
    }
}
```

Spectral Emission

- Dependent on type of fuel
- Dependent on mixture of oxygen
- Also defined as an intensity per wavelength

Butane Spectrum



Stimulus Response

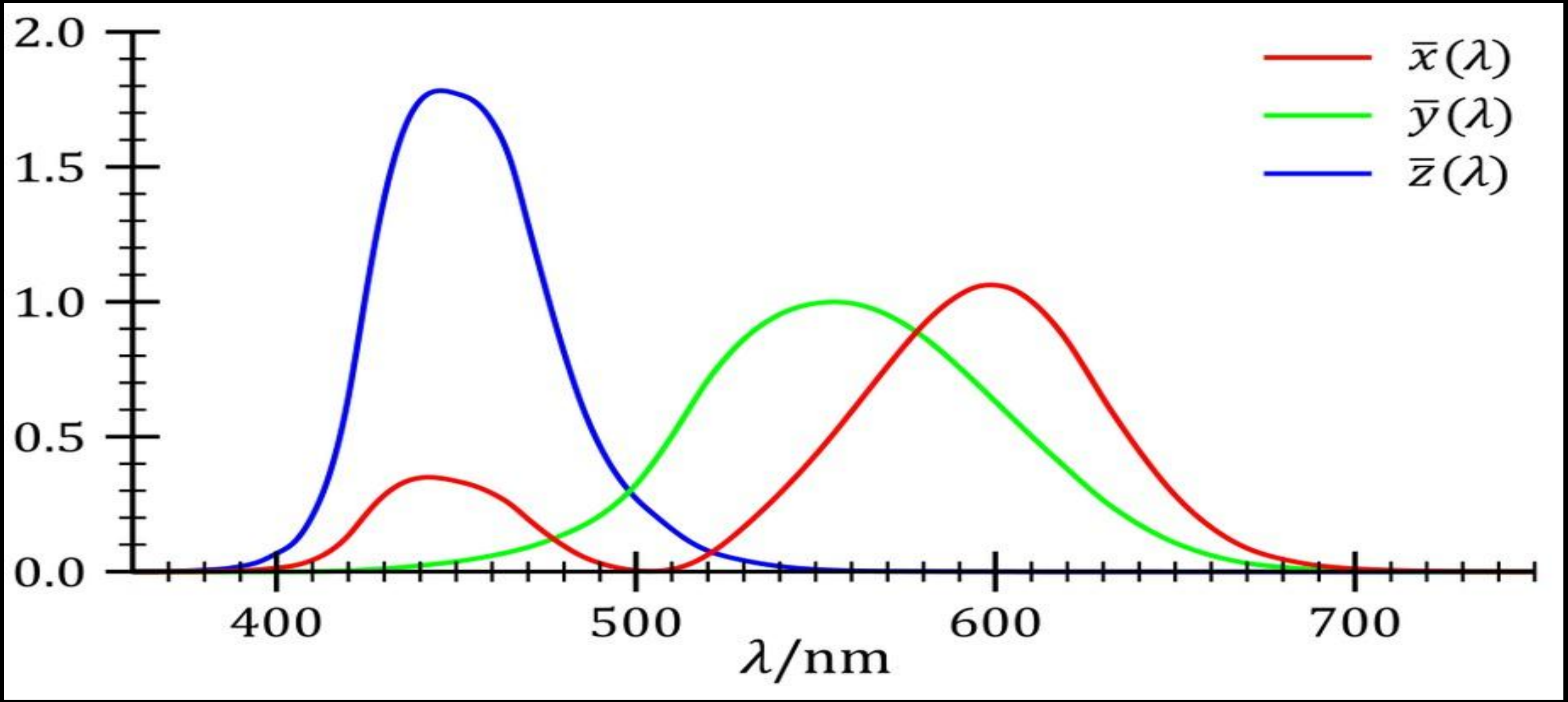
- For a given receptor, a Stimulus Response Curve represents the sensitivity of that receptor to each individual wavelength of light
- The integral of a Stimulus Response Curve with an Emission Spectrum produces a single scalar receptor response to a spectrum of radiation

Tristimulus Response

- Combination of Stimulus Response Curves for a triplet of receptor types
- Human Color Vision composed of three types of cells with different spectral sensitivites, called “cones”. (L, M, S)
- Color Photography created from three types of color sensitive films or sensors, or alternatively three different filters (Technicolor)

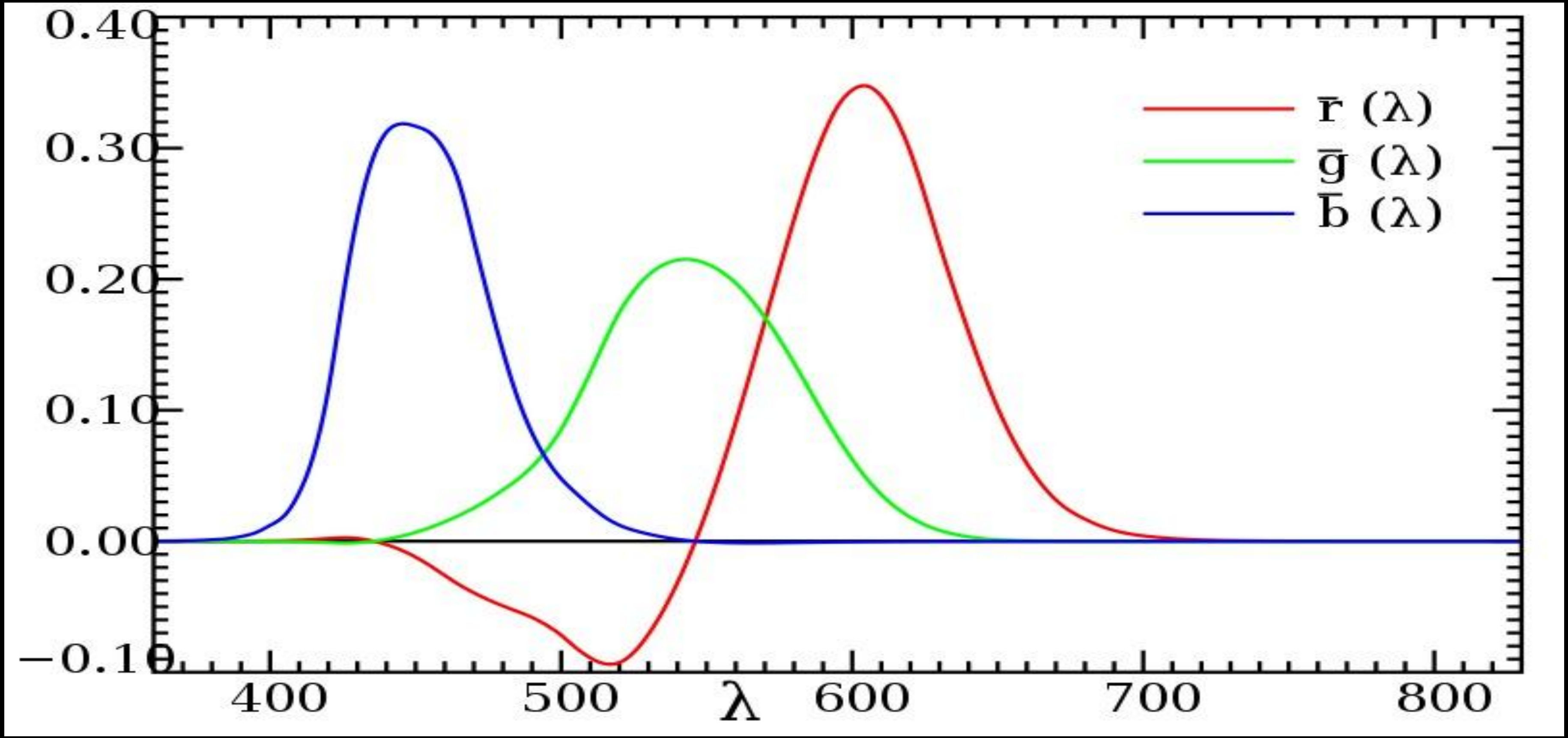
CIE XYZ

- Created in 1931 by International Commission on Illumination



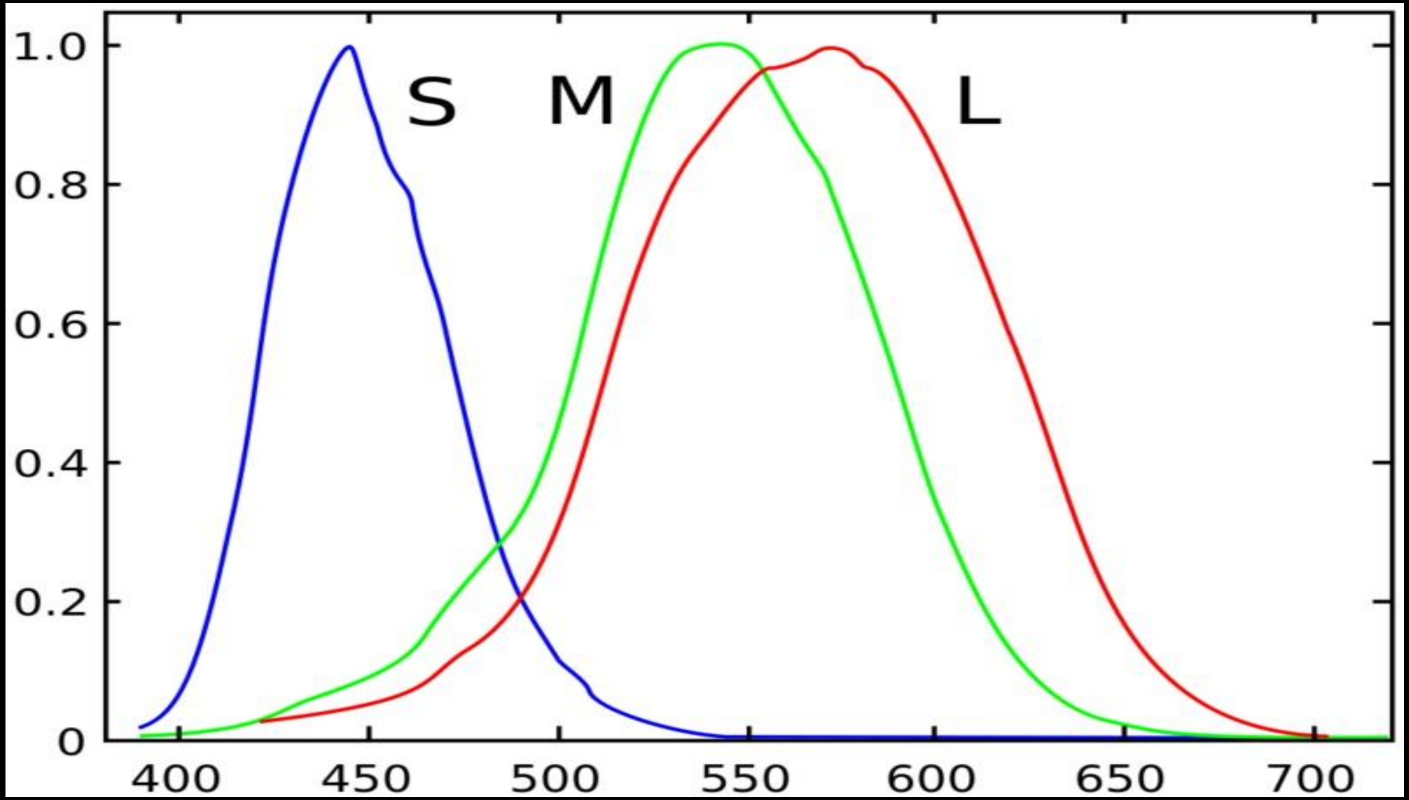
CIE RGB

- RGB curves have negative spectral response



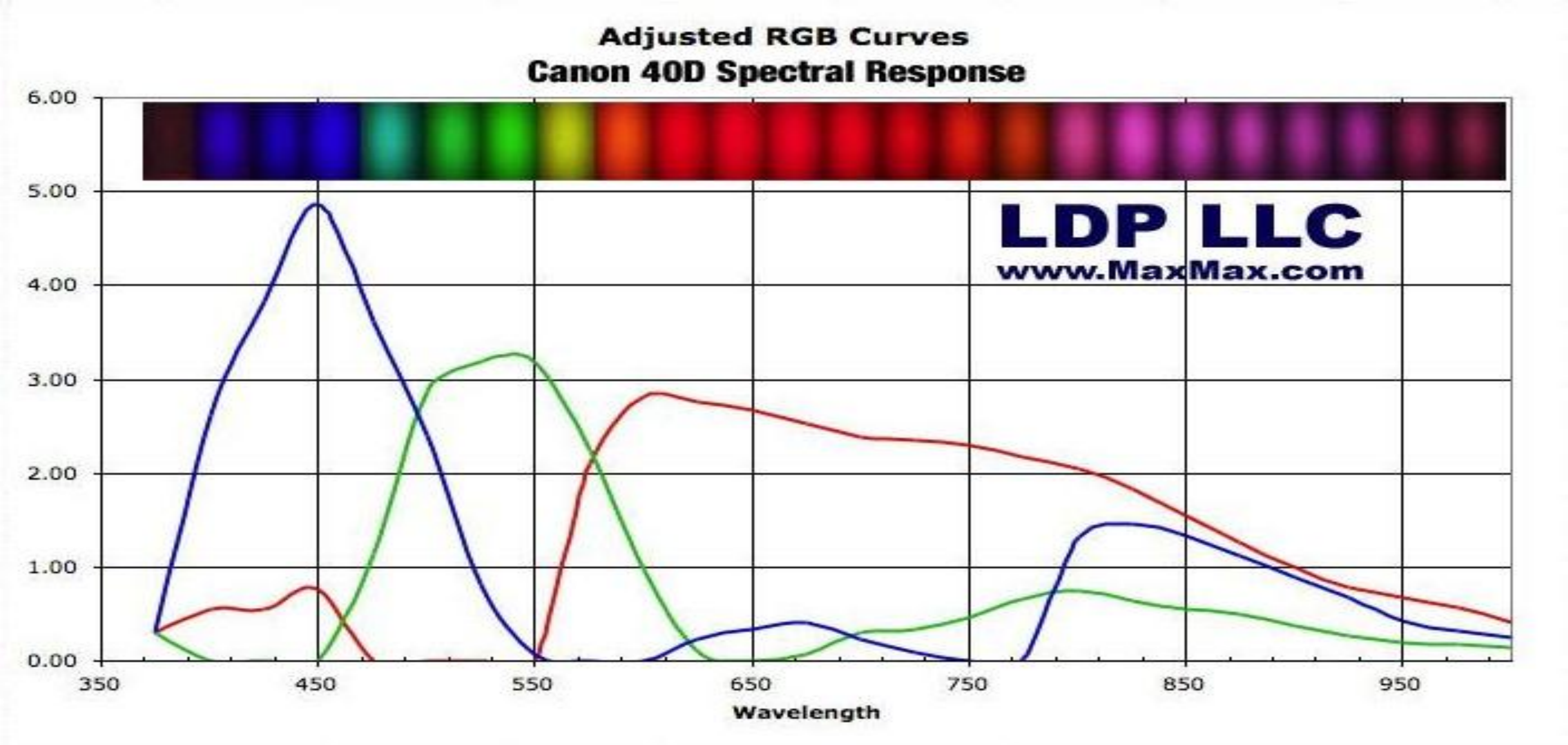
Human Spectral Sensitivity

- Significantly Overlapping in Red and Green



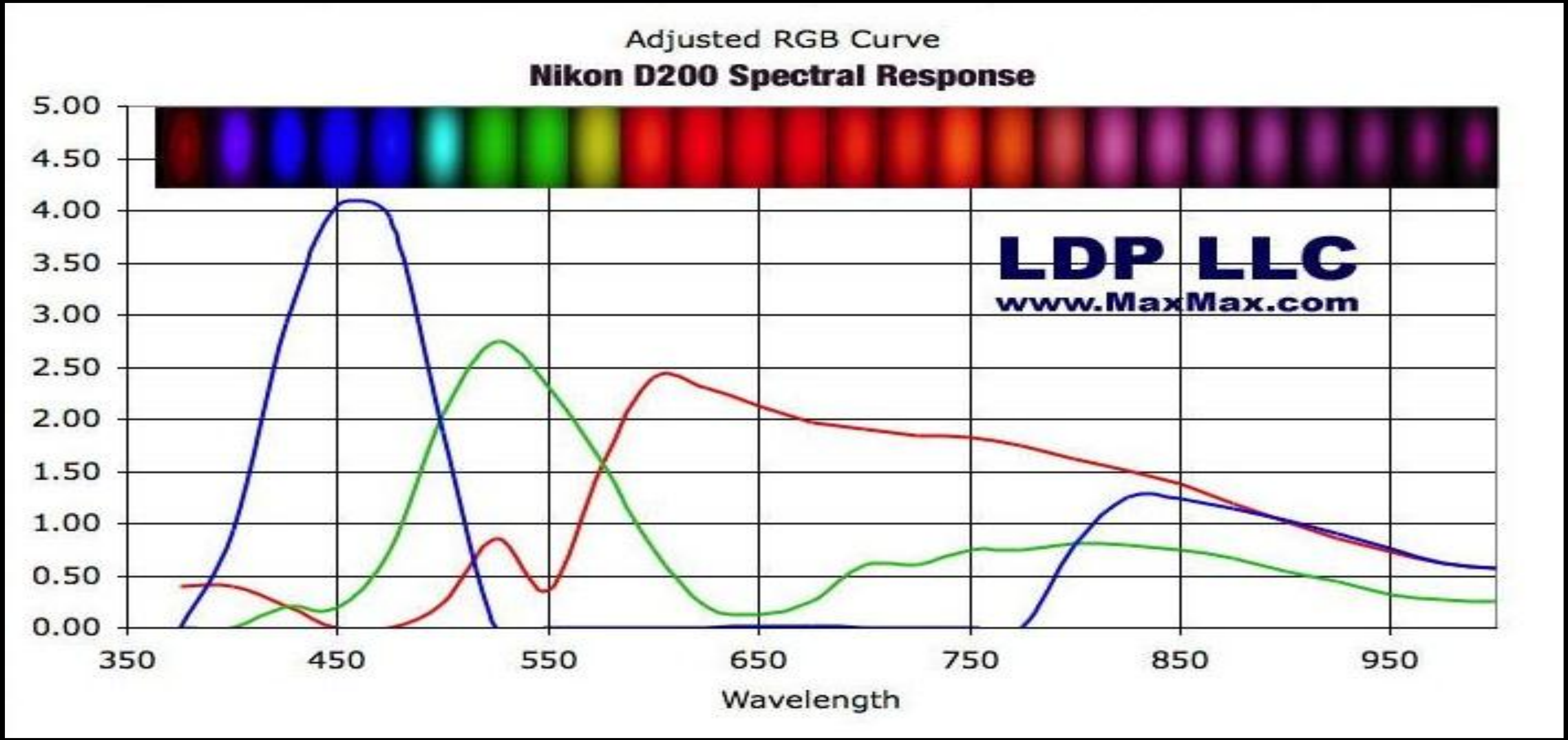
Digital Camera Response

- Significant Infrared Sensitivity without IR Filter



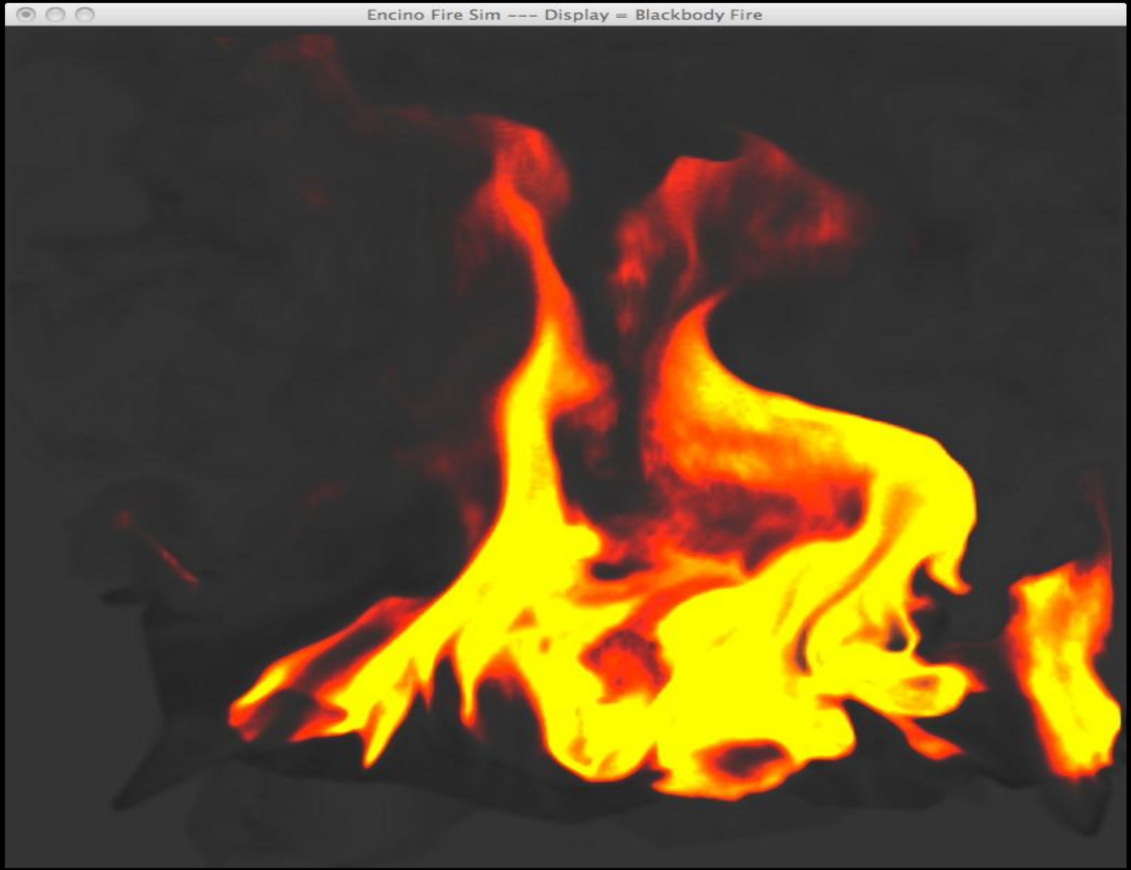
Digital Camera Response

- Significant Infrared Sensitivity without IR Filter



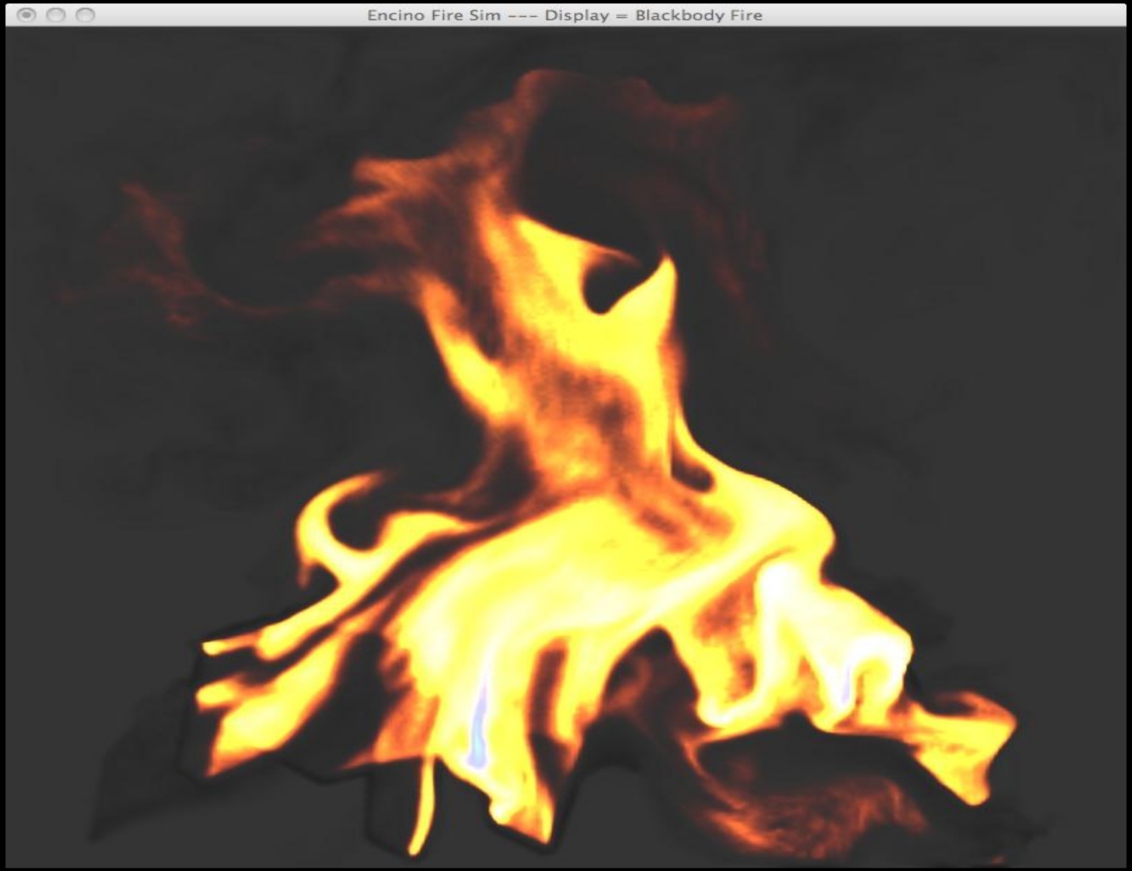
1300K Flame via CIE

- Overly Saturated, “Computer Generated” look.



1300K Flame via cRGB

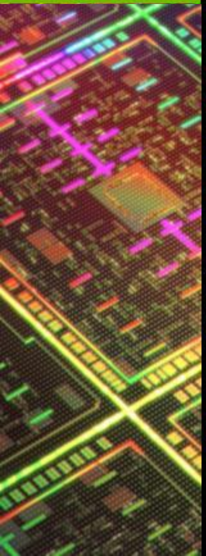
- Properly Balanced Flame Appearance



Questions?

Thanks

- Chris Horvath
- Mark Harris
- Nuttapong Chentanez



References

- Jos Stam, "Stable Fluids", In SIGGRAPH 99 Conference Proceedings, Annual Conference Series, August 1999, 121-128 PDF
- Fast Fluid Dynamics Simulation on the GPU, Mark Harris, GPU Gems
- Real-Time Simulation and Rendering of 3D Fluids, K*eenan Crane, Ignacio Llamas, Sarah Tariq, GPU Gems 3*
- Capturing Thin Features in Smoke Simulations, Siggraph Talk 2011, Magnus Wrenninge, Henrik Falt,Chris Allen, Stephen Marshall PDF