High Dynamic Range Compression on Programmable Graphics Hardware

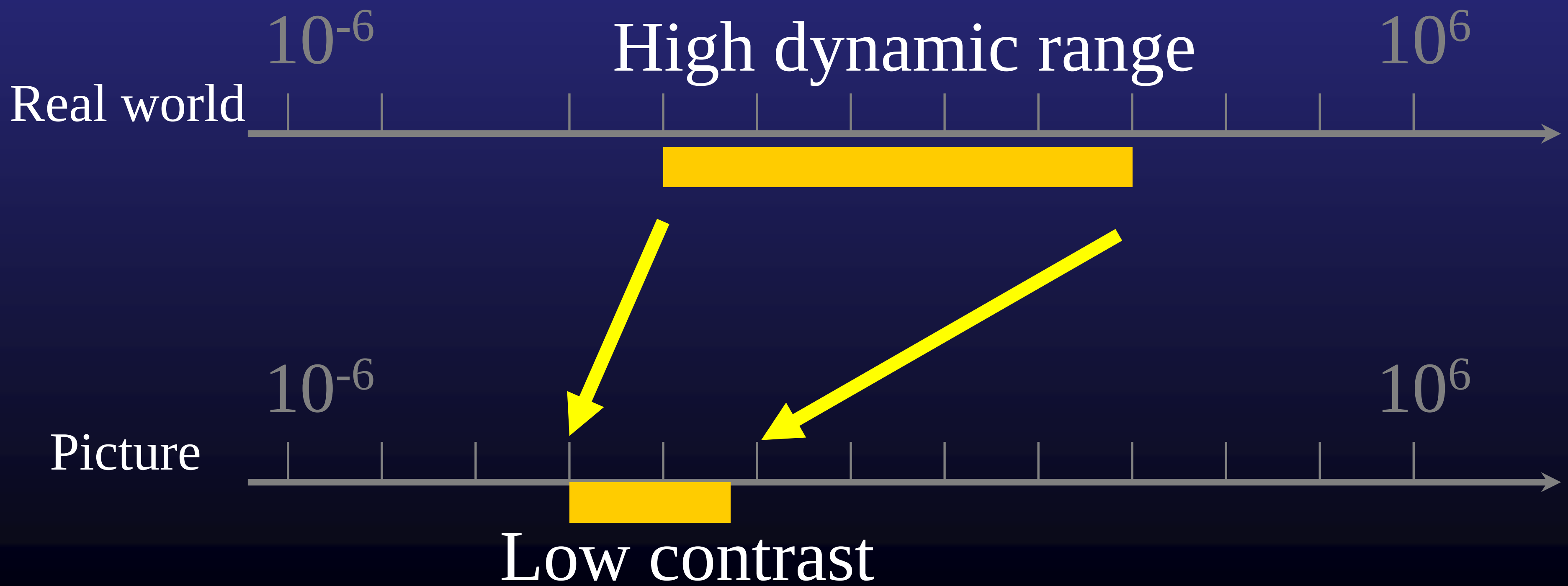
- Overview
- Previous Work
- Implementation
- Conclusion

Overview

- HDRI and Tone Mapping
- Issues we concentrated on

Tone reproduction (Tone mapping)

A method of scaling (or mapping) luminance values in the real world to a displayable range.



Issues we concentrate on

- Important fact

  HVS has a greater sensitivity to relative rather than absolute luminance levels.

- Real Time

  Permit tone reproduction for interactive applications – Implementation on GPU .

- Regardless time dependent

Previous work

- Tone Mapping
  - spatially uniform
  - spatially varying
- Related Works on GPU

Spatially Invariant

- Scale each pixel according to a fixed curve
- Key issue: shape of curve
  - Image-independent curves:
    - Linear scaling, Gamma correction, logarithmic mappings…
  - Image-dependent curves:
    - Histogram equalization
    - Visibility matching tone reproduction (Ward et al. 97)
- Problem: monotonic mapping leads to loss of local contrast!

Spatially Varying

- Scale each pixel by a local average
- Spatially variant tone mapping operators
  - Before 99… (Simple visual models or Simple multi-scale methods)
  - LCIS (Tumblin and Turk 99)
  - Fast Bilateral filter (Durand and Dorsey 2002)
  - Gradient domain (Fattal et al. 2002)
  - Photographic Zone System (Reinhard et al. 02)
  - Perceptual model (Ashikhmin 2002)
- Problem: “halo” artifacts

Gradient Domain HDR Compression

- Keys:
  - High dynamic range results from strong luminance changes
  - Absolute change magnitude is not important
- Method:
  - Examine gradients to identify luminance changes
  - Attenuate high luminance gradients
  - Reconstruct a low-dynamic range image

Gradient Domain HDR Compression

- New method for detail-preserving compression of dynamic range
- Computationally efficient
- Conceptually simple

Gradient-space
[Fattal et al.]

Related Works on GPU

- Conjugate Gradients Solver On GPU
  - Sparse linear system
  - Unstructured Grids
  - Conventional methods and Slow without optimization
- Multigrid Solver On GPU
  - Elliptic PDEs
  - Over Regular Grids

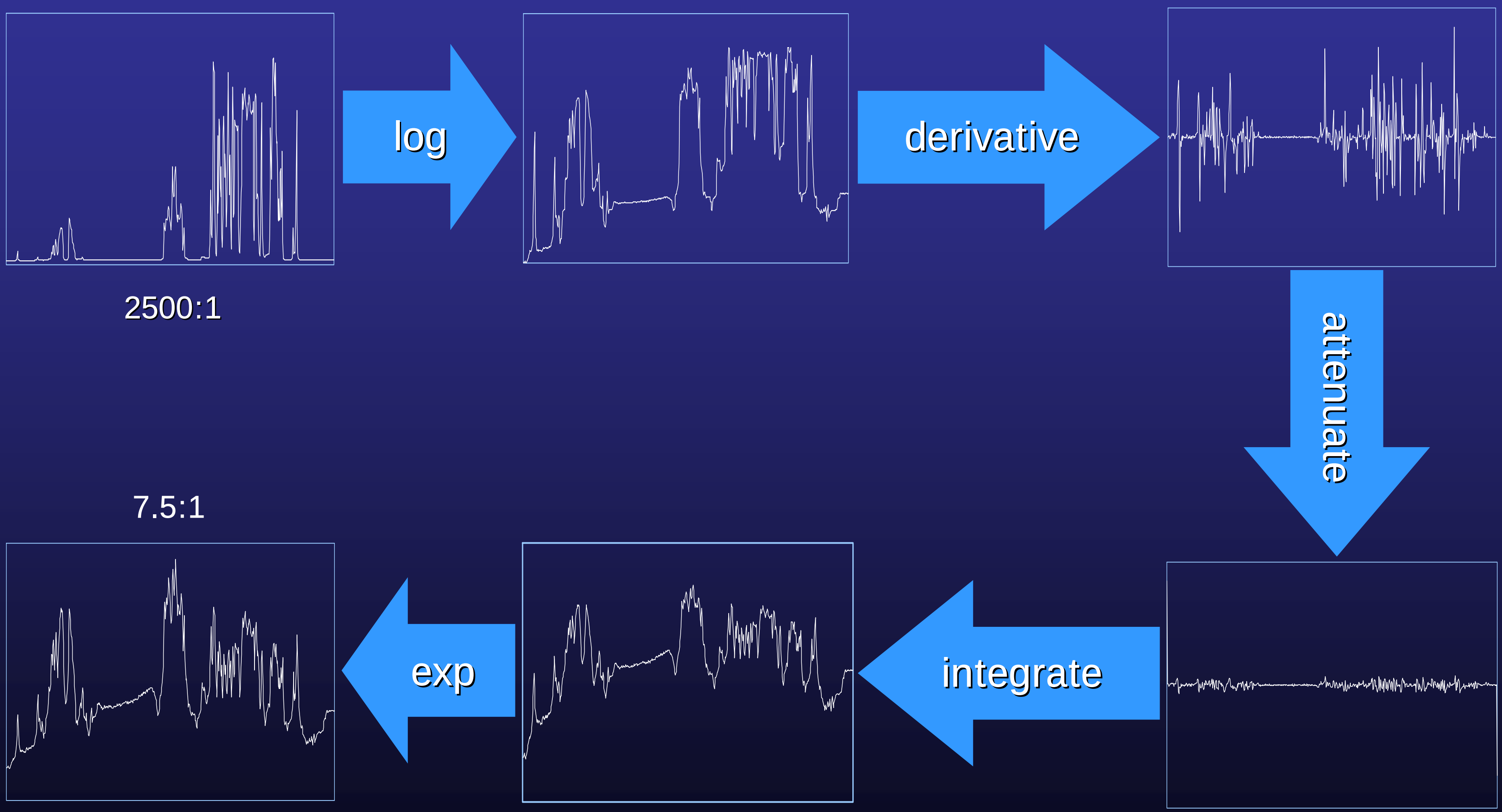
Implementation – GPU GHDRC

- Algorithm
- Implementation On GPU

Algorithm - GHDRC

- Keys:
  - High dynamic range results from strong luminance changes
  - Absolute change magnitude is not important
- Method:
  - Examine gradients to identify luminance changes
  - Attenuate high luminance gradients
  - Reconstruct a low-dynamic range image

The Method in 1D



The Method in 2D

- Given: a log-luminance image $H(x,y)$
- Compute an attenuation map $\Phi\left(\left\|\nabla H\right\|\right)$
- Compute an attenuated gradient field G:

$$G\left(x,y\right)=\nabla H\left(x,y\right)\cdot\Phi\left(\left\|\nabla H\right\|\right)$$

- Problem: G is not integrable!

Solution

- Look for image I with gradient closest to G in the least squares sense.

- I minimizes the integral: $\iint F(\nabla I, G)\,dx\,dy$

$$F(\nabla I, G) = \|\nabla I - G\|^2 = \left(\frac{\partial I}{\partial x} - G_x\right)^2 + \left(\frac{\partial I}{\partial y} - G_y\right)^2$$

Euler-Lagrange Equation

- I must satisfy: $$\frac{\partial F}{\partial I} - \frac{d}{dx}\frac{\partial F}{\partial I_x} - \frac{d}{dy}\frac{\partial F}{\partial I_y} = 0$$
- Substituting F we get (Poisson equation):

$$\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} = \frac{\partial G_x}{\partial x} + \frac{\partial G_y}{\partial y}$$

$$\nabla^2 I = \operatorname{div} G$$

Linear system of equations

- Standard finite differences

$$\nabla^2 I(x,y) \approx I(x+1,y)+I(x-1,y)+I(x,y+1)+I(x,y-1)-4I(x,y)$$

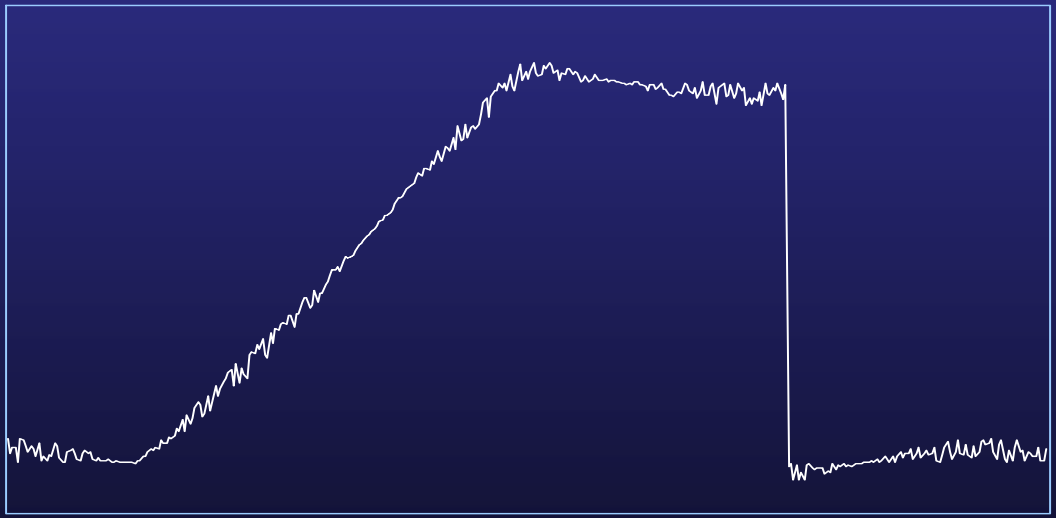
$$\nabla H(x,y) \approx (H(x+1,y)-H(x,y), H(x,y+1)-H(x,y))$$

$$divG \approx G_x(x,y)-G_x(x-1,y)+G_y(x,y)-G_y(x,y-1)$$

- Solve Poisson equation using the *<u>Optimized Conjugate Gradient Method</u>*

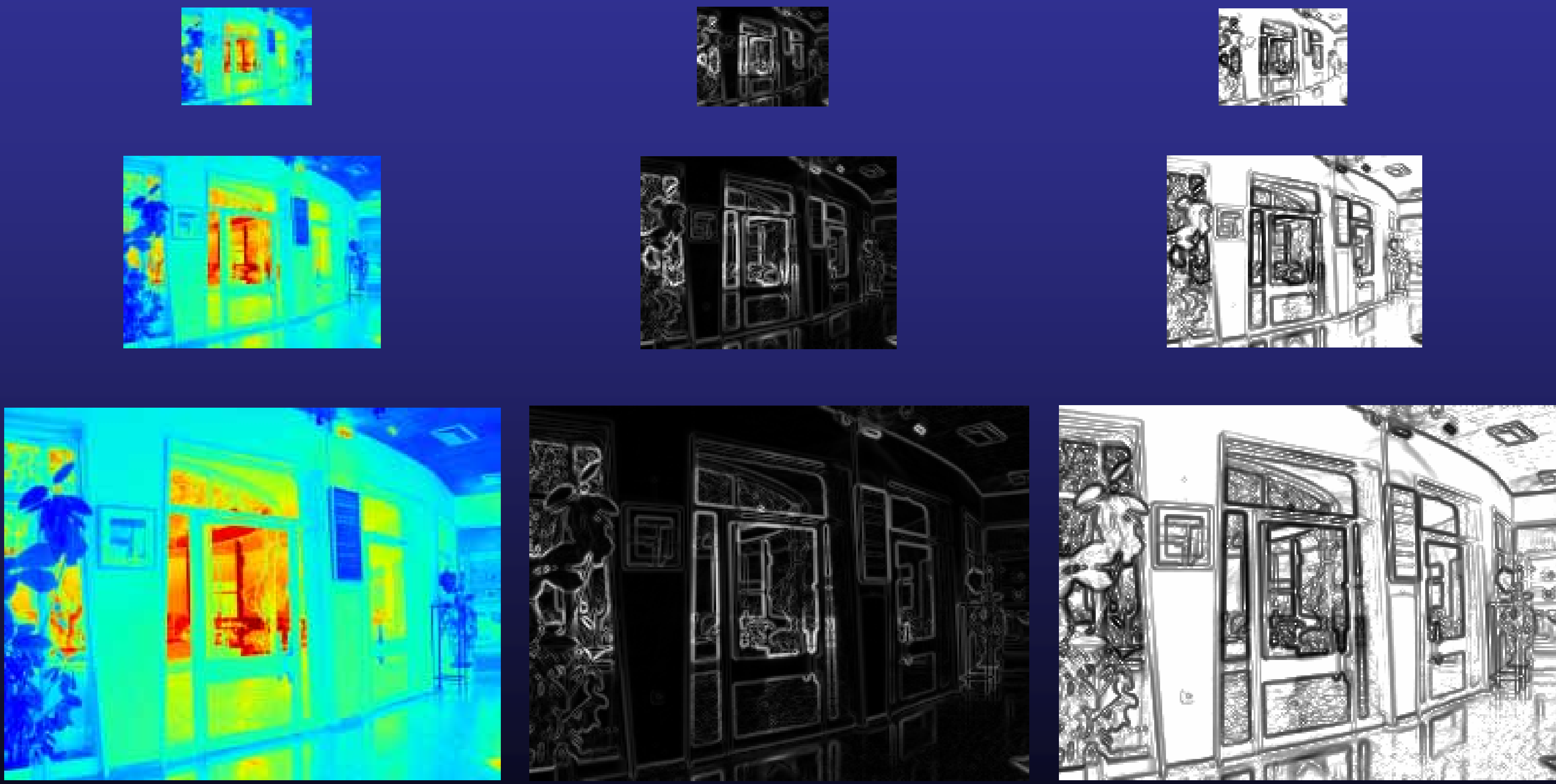
Gradient attenuation function

- Strong luminance changes may occur at different rates:



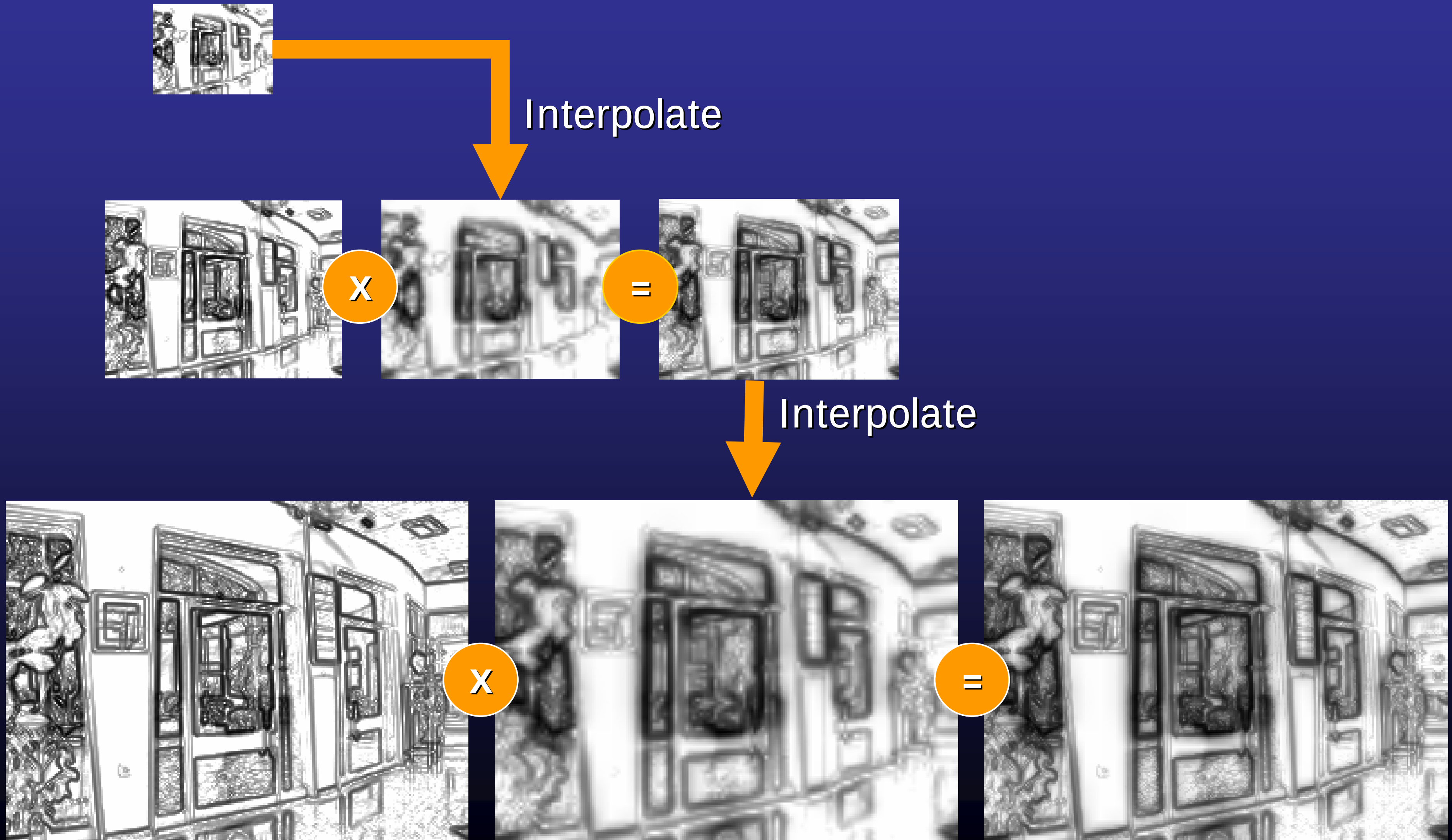
- Must examine gradients at multiple scales!

Multiscale Gradient Attenuation



log(Luminance) Gradient magnitude Attenuation map

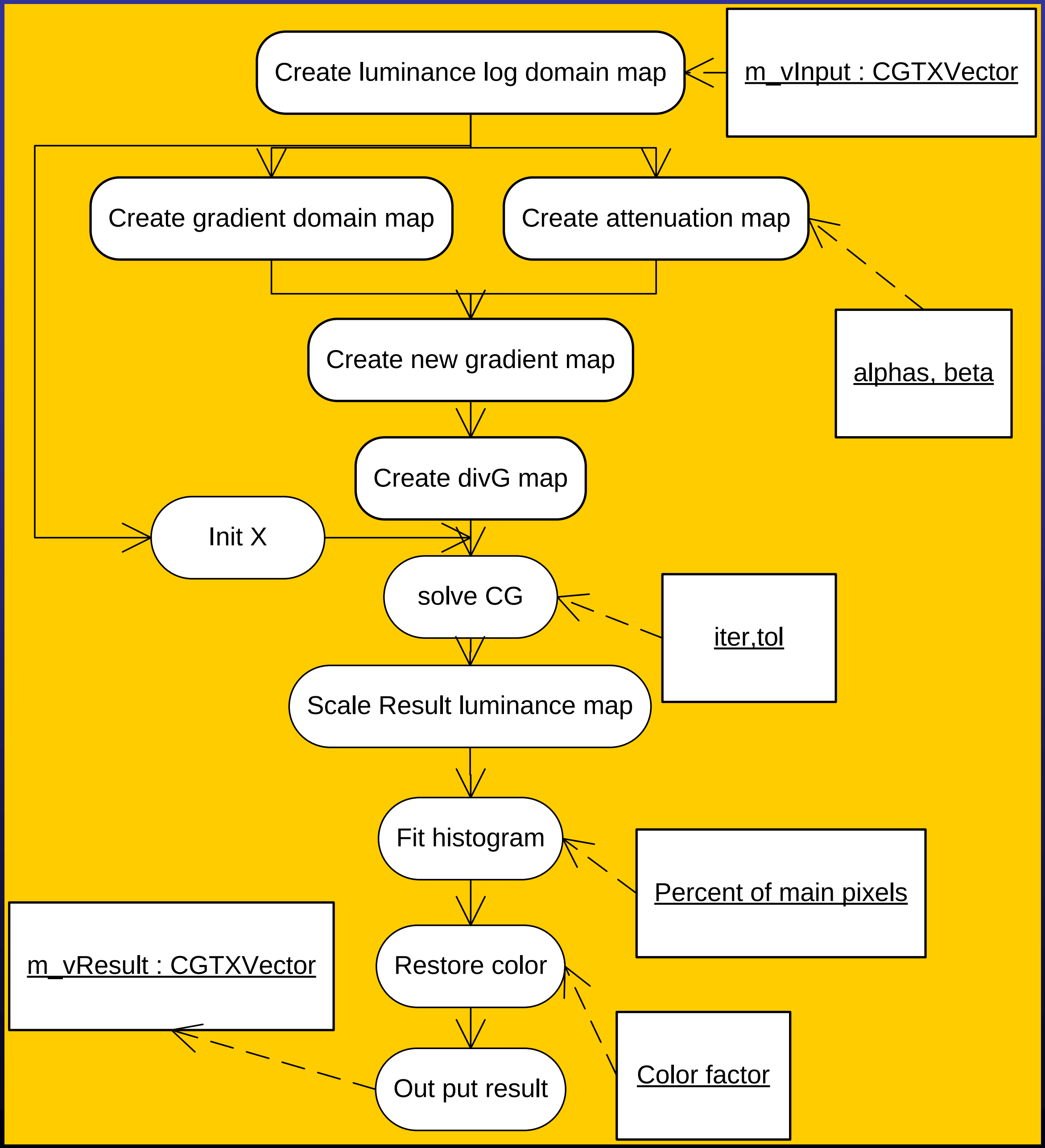
Multiscale Gradient Attenuation



Final Gradient Attenuation Map

Implementation - Algorithm

- GHDRC Flow



Optimized Conjugate Gradient Method

$$A \cdot x = b$$

Initial guess x_1

$$r_1 = b - A \cdot x_1$$

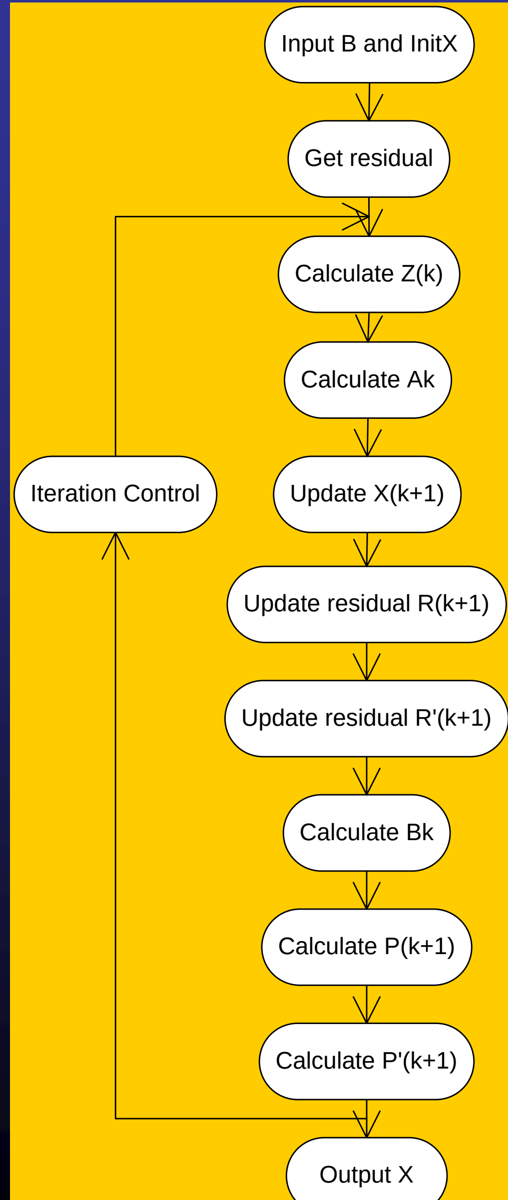
Iterations

$$x_{k+1} = x_k + \alpha_k p_k$$

Implementation - Algorithm

Common CG Iteration

- Sparse matrix A is Symmetric
- A' is a preconditioner



$$r_1 = b - A \cdot x_1, \bar{r}_1 = A \cdot r_1$$

$$A' \cdot z_k = r_k, A'^T \cdot \bar{z}_k = \bar{r}_k$$

$$\alpha_k = \frac{\bar{r}_k \cdot z_k}{\bar{p}_k \cdot A \cdot p_k}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k \cdot A \cdot p_k$$

$$\bar{r}_{k+1} = \bar{r}_k - \alpha_k \cdot A^T \cdot \bar{p}_k$$

$$\beta_k = \frac{\bar{r}_{k+1} \cdot z_{k+1}}{\bar{r}_k \cdot z_k}$$

$$p_{k+1} = z_{k+1} + \beta_k \cdot p_k$$

$$\bar{p}_{k+1} = \bar{z}_{k+1} + \beta_k \cdot \bar{p}_k$$

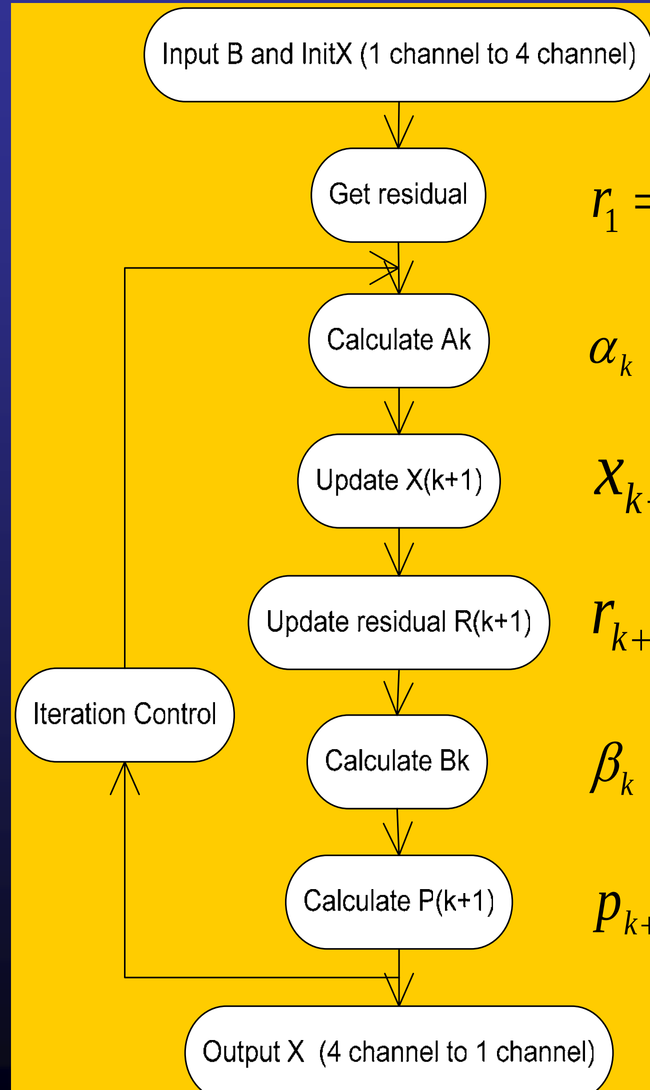
Faster CG Iteration

- Sparse matrix A is positive definite as well as symmetric
- Simplify vector operations
- Optimized operations on A‘

$$\begin{cases} \alpha_k = (-1/4) \cdot \dfrac{r_k \cdot r_k}{p_k \cdot A \cdot p_k} \\ \beta_k = \left|r_{k+1}\right|^2 / \left|r_k\right|^2 \\ r_{k+1} = r_k - \alpha_k \cdot A \cdot p_k \\ p_{k+1} = (-1/4) \cdot r_k + \beta_k \cdot p_k \end{cases}$$

Faster CG Iteration

- Can not break down (in theory!)
- A’ is the diagonal part of A, the rate of convergence is higher.



$$r_1 = b - A \cdot x_1$$

$$\alpha_k = (-1/4) \cdot \frac{r_k \cdot r_k}{p_k \cdot A \cdot p_k}$$

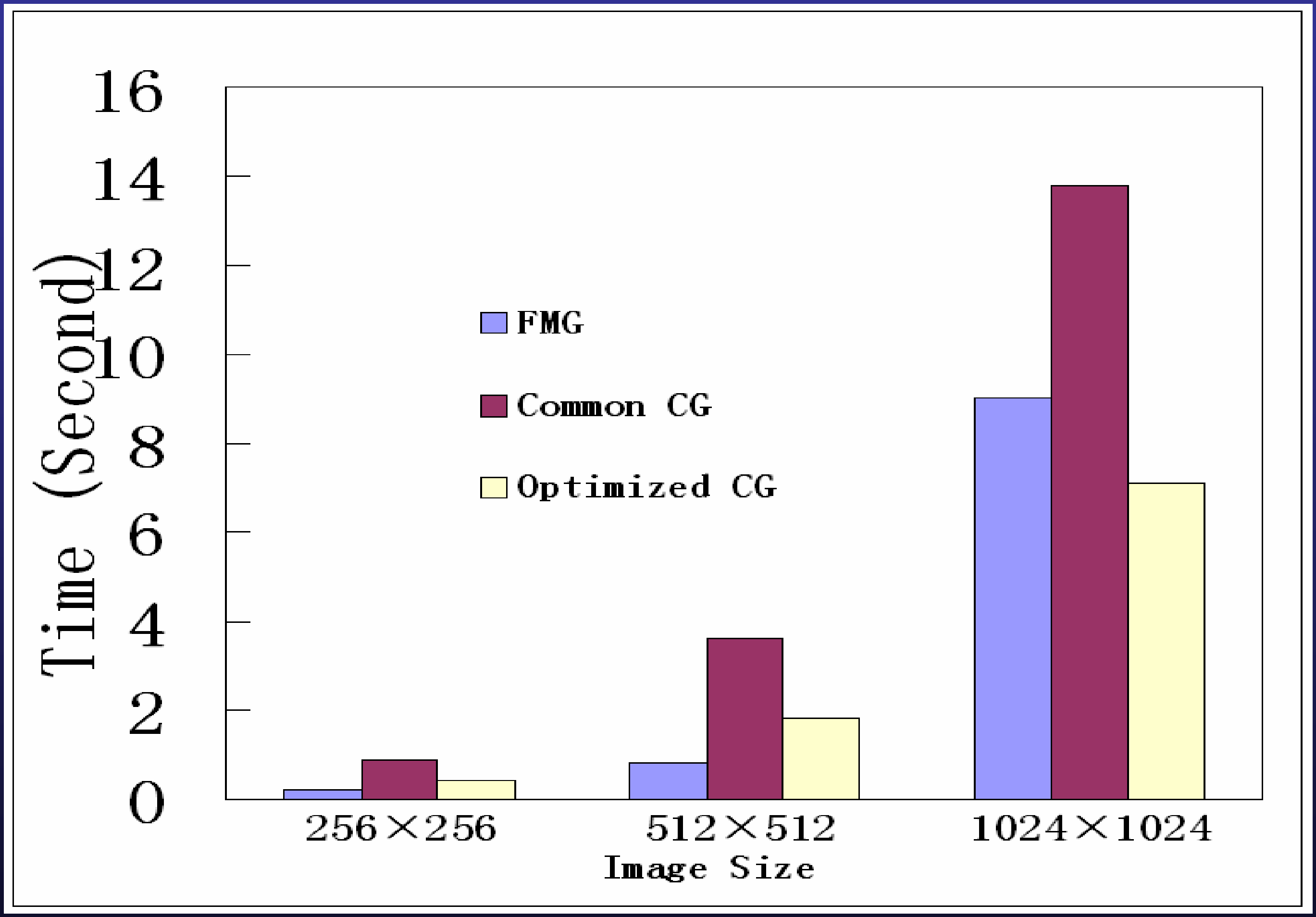
$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k \cdot A \cdot p_k$$

$$\beta_k = \frac{\left|r_{k+1}\right|^2}{\left|r_k\right|^2}$$

$$p_{k+1} = (-1/4) \cdot r_{k+1} + \beta_k \cdot p_k$$

Faster CG Iteration

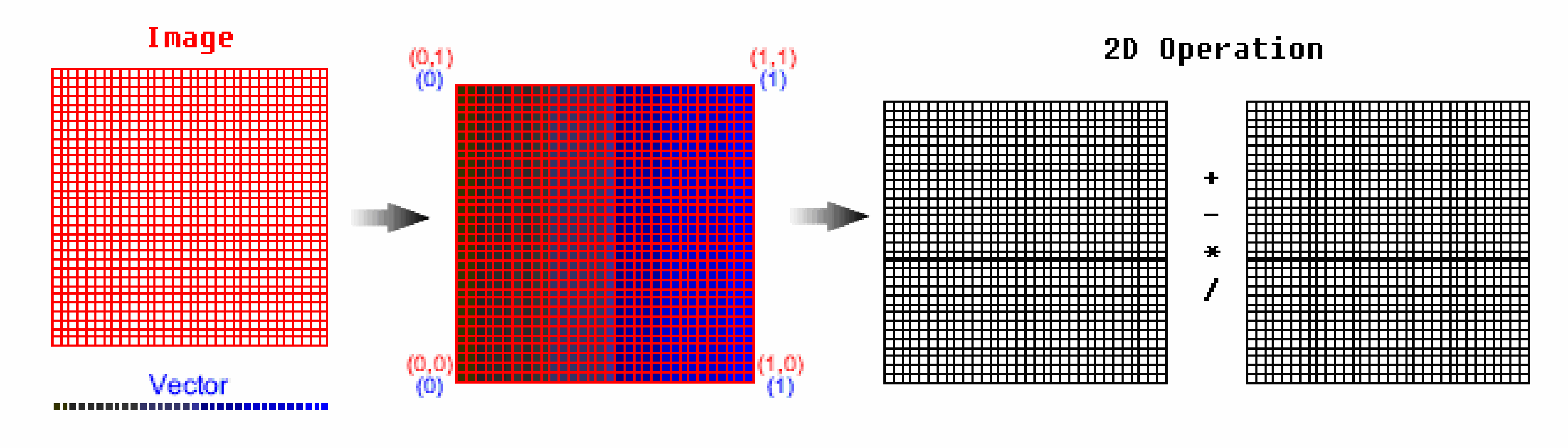


Run CG iteration 100 times on CPU（ α =0.1, β =0.88)

Implementation On GPU

- Data Structure
  - Texture
- Basic Operations
  - Vector
  - Sparse matrix A
- Optimizations
- Performance

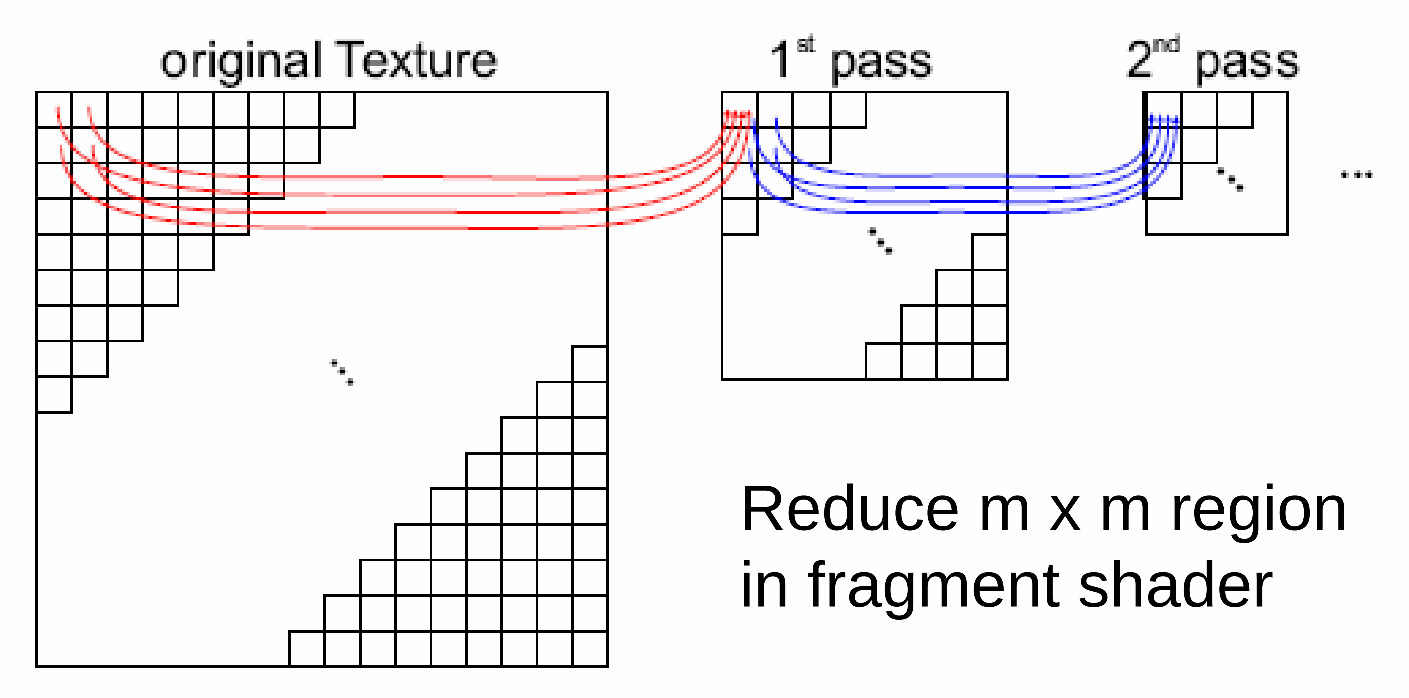
Data Structure -Texture



- All operations for vectors or matrices are replaced by Texture rendering with shaders

Basic Operations

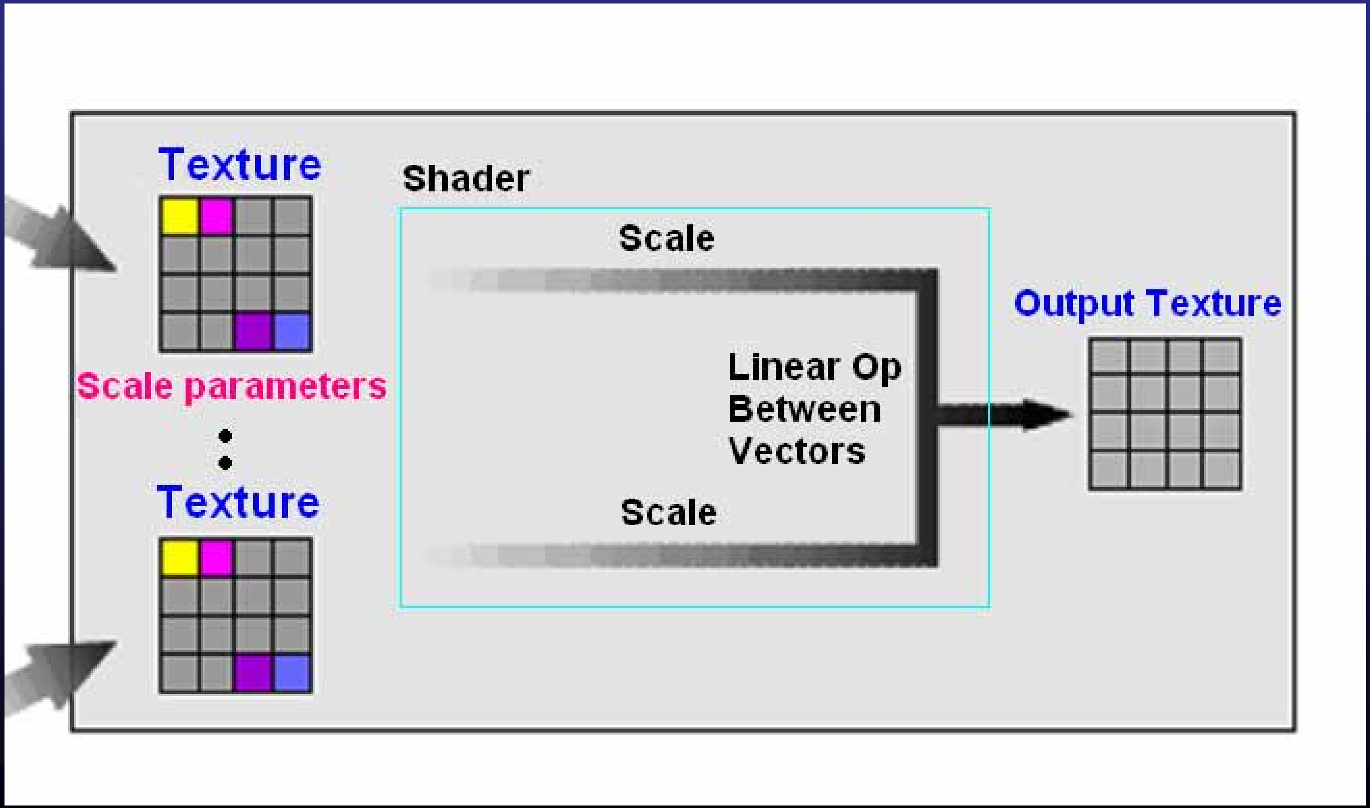
- Vector Dot Product



Basic Operations

- Linear Combination of Vectors

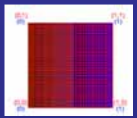
$$\vec{V} = s_1 \cdot \vec{V_1} + s_2 \cdot \vec{V_2}$$



Basic Operations

- Matrix A multiply vector

$$\vec{V} => I$$

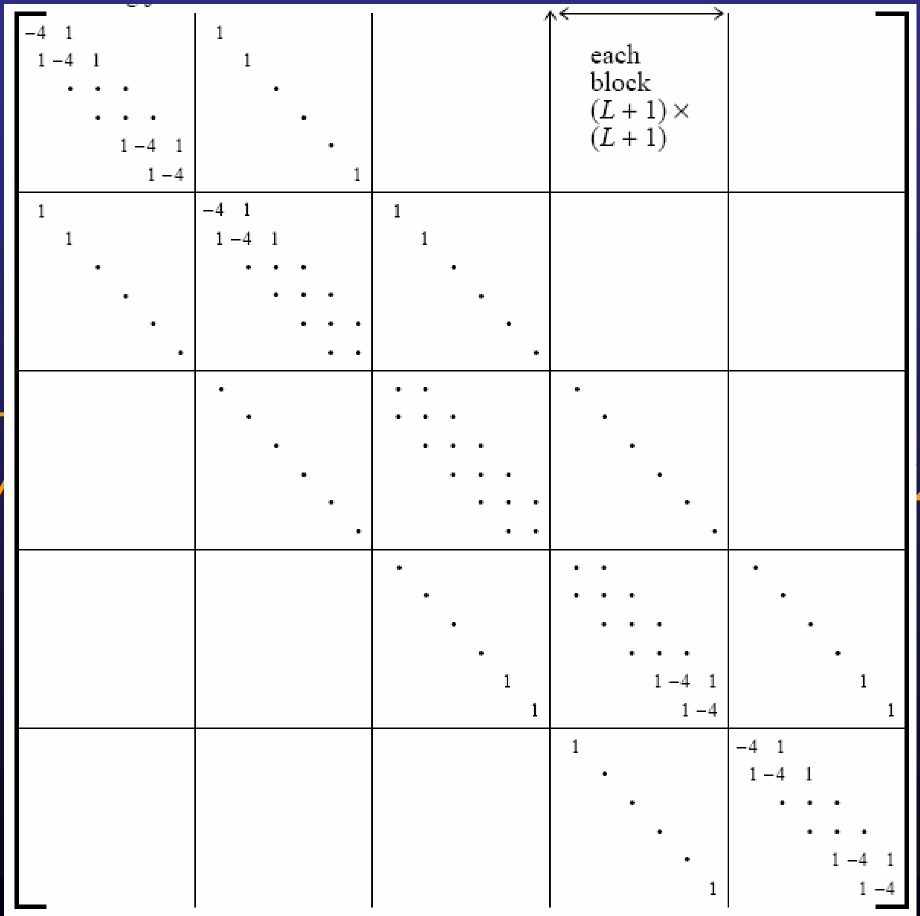


$$A\cdot\vec{V} = (-4)\cdot\vec{V}$$

$$+\vec{V}_{+1} + \vec{V}_{-1}$$

$$+\vec{V}_{+N} + \vec{V}_{-N}$$

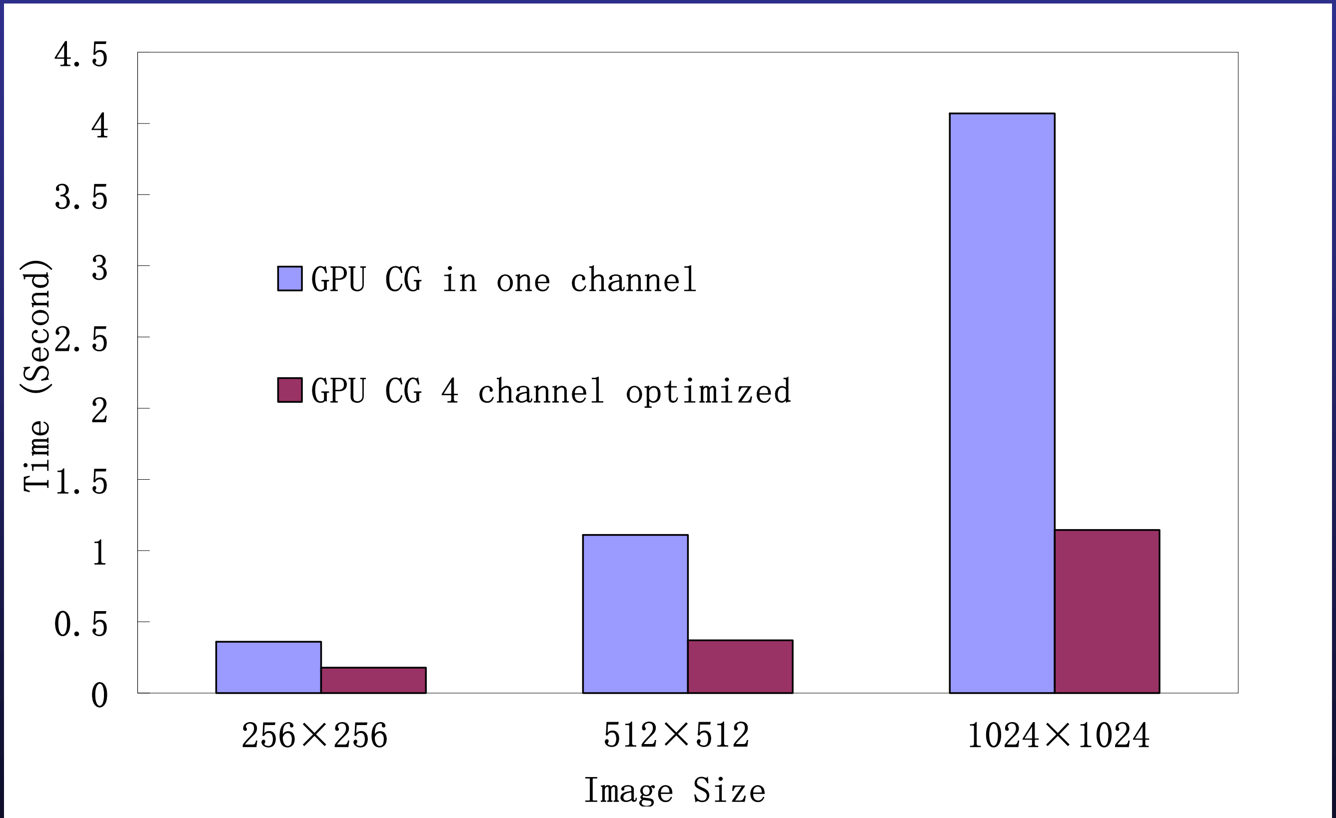
$$A\cdot\vec{V} \ldots x-1,y)+I(x,y+1)+I(x,y-1)]$$



V_i : Shifted Vector

Optimizations

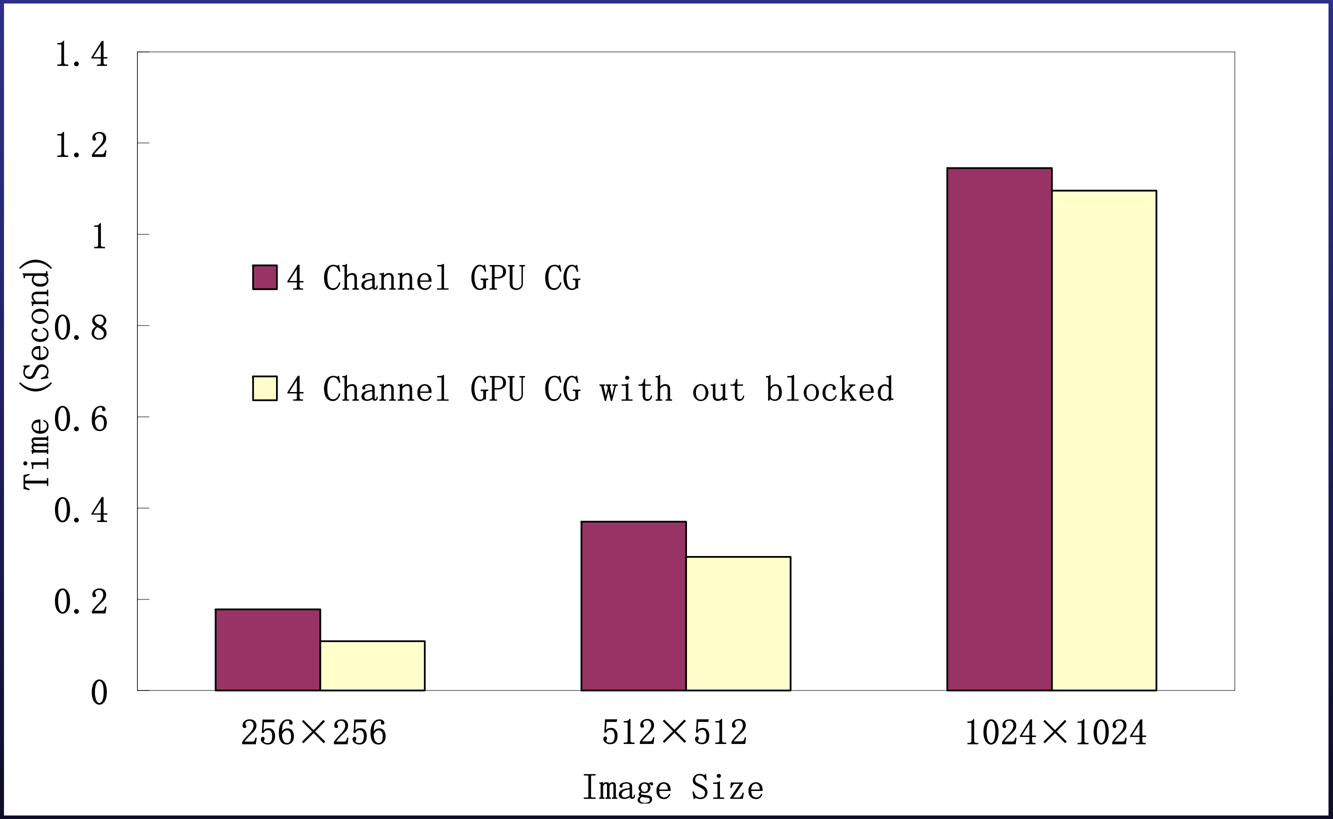
- Full RGBA Channel (IEEE A32B32G32R32F)



Run CG iteration 100 times on GPU（ α =0.1, β =0.88)

Optimizations

- Without Locking data from Texture



Run CG iteration 100 times on GPU（α =0.1, β =0.88)

Performance

- Measured on 2.8 GHz P4, ATI 9800XT:
  - 512 x 512: 0.4 sec
  - 1024 x 1024: 1.5 sec

Conclusion

- Iterate 500 times 1.57s

Conclusion

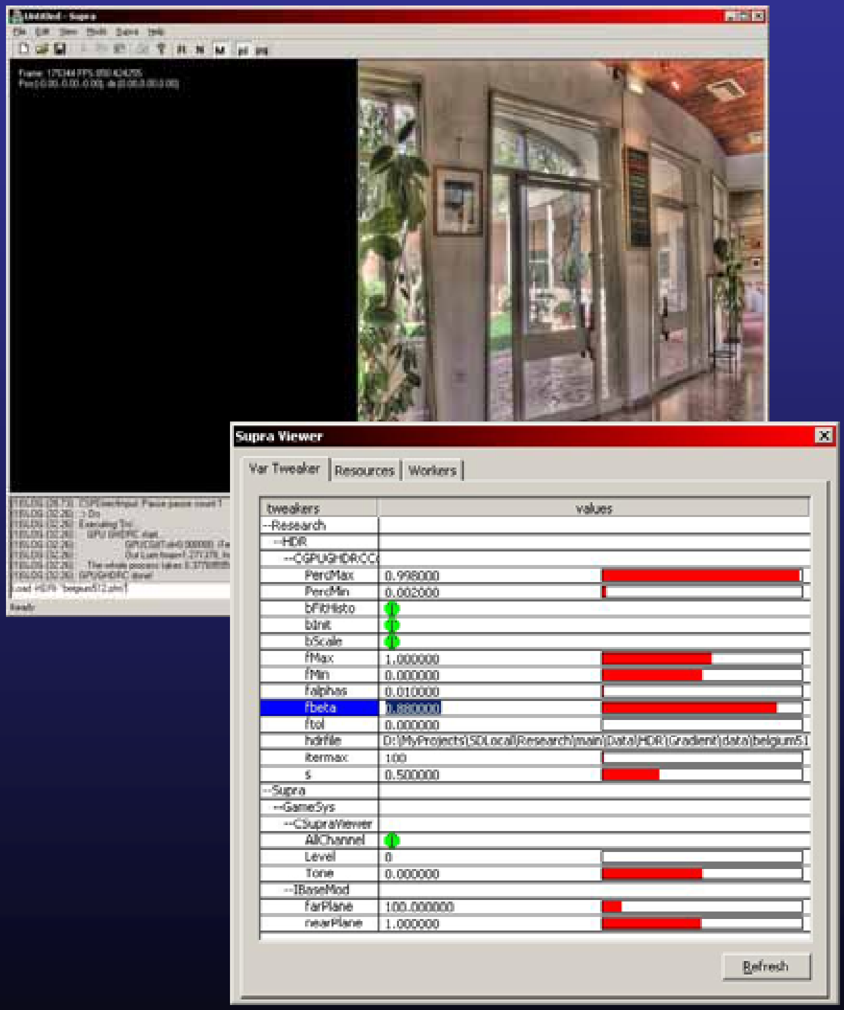
- Iterate 100 times 0.4s

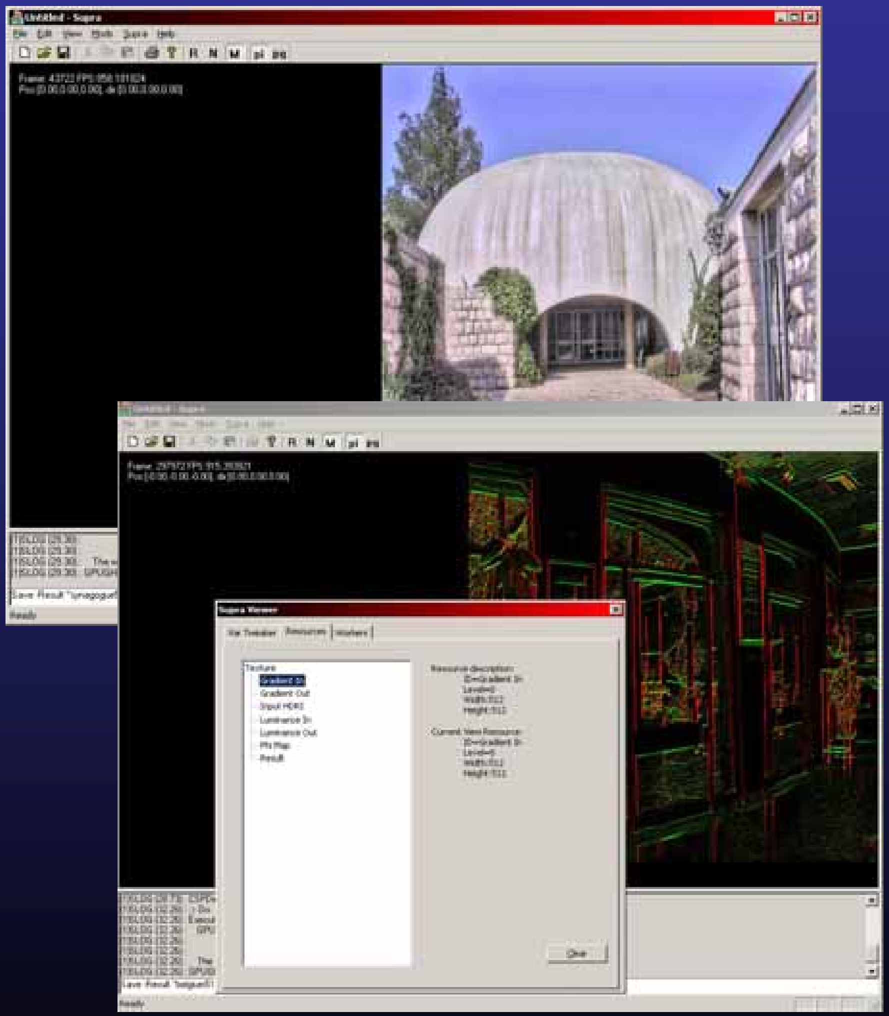
Conclusion

- Optimized shaders
- 4 Channel On Whole Method

Conclusion

GHDRC Tool and Lib





Thanks