Structured Light Based 3D Reconstruction With Priors

O.Hamud ,B.Matok - under supervision of G. Rosman

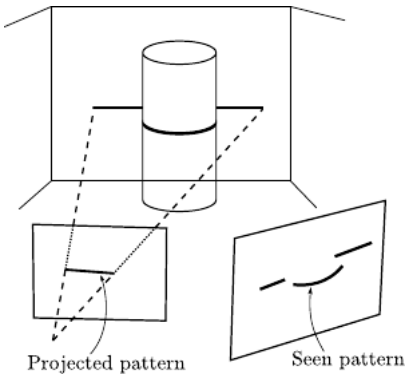
July 20, 2013

Part I
Introduction to 3D Reconstruction Technique

1 Structured Light

"Structured Light" is used to describe any method that:

- Emits light based patterns on an object.
- Takes pictures of the object.
- Extracts the 3D structure of the scene based on the how the pattern interacts with the object.

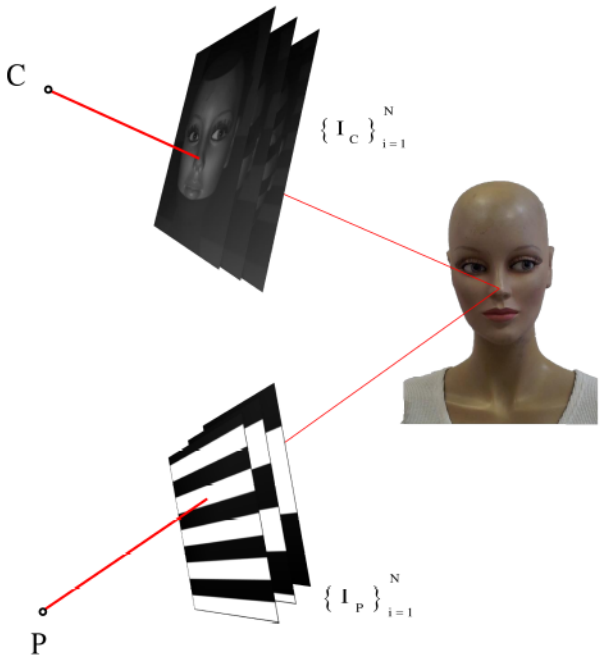


[4]

2 Our Apparatus

We used a structured light system with the following components.

- A single projector to project Grey coded patterns on an object.
- A single camera in a single position, taking multiple pictures (one per projected pattern).
- CUDA enabled platform to do heavy calculation that outputs the 3D structure of the object.



qualitative description of the setup

2.1 High level algorithm

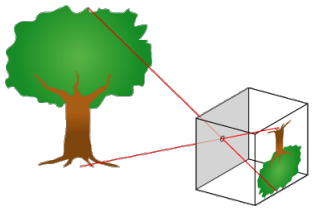
For each pixel, we guessed the depth of the part of the object (called back-projection), and calculated a relative probability term, after guessing many such values with different probabilities we picked the most probable solution based on our probability model.

1. for each camera pixel x
   (a) for each plane, at distance z from the camera do
   (b) find the point X in the plane, that projecting it into the screen gives x.
      i. project X into the projector screen, mark the point $\Pi(x, z)$
      ii. mark the color of pixel x in the i-th picture as $I_C^{(i)}(x)$
      iii. mark the color of pixel $\Pi(x, z)$ in the projector's i-th image as $I_P^{(i)}(\Pi(x, z))$
      iv. given $\{I_C^{(i)}(x)\}_{i=1}^N$ and $\{I_P^{(i)}(\Pi(x, z))\}_{i=1}^N$, calculate the probability that for each i, pixel $I_C^{(i)}(x)$ is filming the object being projected with $I_P^{(i)}(\Pi(x, z))$, mark this as $p(x, z)$
   (c) for pixel "x", return the most probable "z", for which $p(x, z)$ is highest.

In the following sections we will explain in detail how to perform each step.

Part II
Reconstruction Model

3 Pinhole Camera Model[8]



The camera and the projector we used, are built from a screen/sensor and a lens.
However, our implementation is based on an approximated model called pin-hole model.
The pin-hole model approximates effects of the camera's focus and it's lens thickness.
Based on this model, taking a picture involeves projecting a 3D object into a 2D image by extending lines from each point in the object into a screen,
passing through the pin-hole (camera center).

The pin-hole model allows a simple mathematical description of the projection operation using a 3x4 matrix.
Acquiring this matrix requires a calibration process which is not in the scope of our project.
Our algorithm assumes a calibration which is available as an input.

3.1 Mathematical Model for Camera And Projector

3.1.1 Homogeneous Coordinates

Homogeneous coordinates are used mainly because it simplifies the projection equations.
It is a way of representing 2D or 3D points as follows:
2D:(x, y) is represented in Homogeneous Coordinates by : $w\,(x, y, 1)$for each $w \neq 0$.
3D:(x, y, z) is represented in Homogeneous Coordinates by : $t\,(x, y, z, 1)$for each $t \neq 0$.

- Notice that regular coordinates are not uniquely represented in homogeneous coordinates - $(xw, yw, w) \Leftrightarrow \underbrace{(x, y, 1)}_{w=1}$
- We notice that $(x, y) \Leftrightarrow (xw, yw, w) \Leftrightarrow (x, y, 1)$ - so dividing by w gives us the normalized representation.

3.1.2 Projection

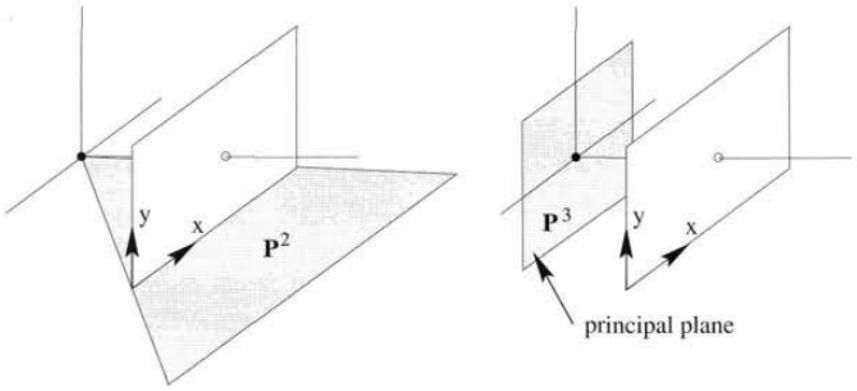
As mentioned above, the projection matrix can be described by a 3x4 matrix:
given a projection matrix P, and a 3D point $\mathbf{X}$ in Homogeneous Coordinates: $\mathbf{X} = (X, Y, Z, T)^T$.
the projection process is described as follows:
multiplying P with 3D point $\mathbf{X}$ gives $\mathbf{x}$, a 2D point on the camera screen:

$$P\mathbf{X} = w\,(x, y, 1)^T = \mathbf{x}$$

3.1.3 Matrix Structure

$$P = \begin{bmatrix} p_1^1 & p_2^1 & p_3^1 & p_4^1 \\ p_1^2 & p_2^2 & p_3^2 & p_4^2 \\ p_1^3 & p_2^3 & p_3^3 & p_4^3 \end{bmatrix} \triangleq \left[\begin{array}{ccc|c} \mathbf{m}^1 & & & p_4^1 \\ \mathbf{m}^2 & & & p_4^2 \\ \mathbf{m}^3 & & & p_4^3 \end{array}\right] \triangleq \begin{bmatrix} \mathbf{P}^1 \\ \mathbf{P}^2 \\ \mathbf{P}^3 \end{bmatrix} \quad ; \quad \mathbf{M} = \begin{bmatrix} \mathbf{m}^1 \\ \mathbf{m}^2 \\ \mathbf{m}^3 \end{bmatrix}$$

where $\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3$ are 1x4 row vectors representing different planes.
and $\mathbf{m}^1, \mathbf{m}^2, \mathbf{m}^3$ are 1x3 row vectors representing the normals to the planes represented by $\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3$.



The camera center (pin-hole) is the intersection point of all 3 planes represented by $\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3$.
$\mathbf{P^3}$−represents the principle plane, which is a plane parallel to the image plane.
$\mathbf{P^2}$−represent the plane which contains all the 3D points that are projected into $w\,(0, y, 1)$.
$\mathbf{P^1}$−represent the plane which contains all the 3D points that are projected into $w\,(x, 0, 1)$.

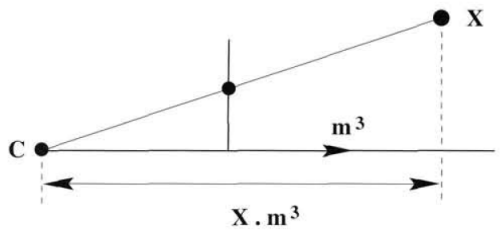
3.1.4 Camera Center

the camera center C is the point for which $PC = 0$, is obtained numerically by marking the column elements of P, $P = [p_1, p_2, p_3, p_4]$ then calculating:

$$\begin{array}{ll} X = det([p_2, p_3, p_4]) & Y = -det([p_1, p_3, p_4]) \\ Z = det([p_1, p_2, p_4]) & T = -det([p_1, p_2, p_3]) \end{array} \tag{1}$$

3.1.5 Back-projection

We remember that projecting a 3D point into the 2D screen is performed by passing a line from the 3D point to the camera center, and finding the line's intersection point with the image screen.
Therefore, back-projection is the reverse process; passing a line from the camera center to a 2D point on the screen to any 3D point that would have been projected to that point (all on the same line)



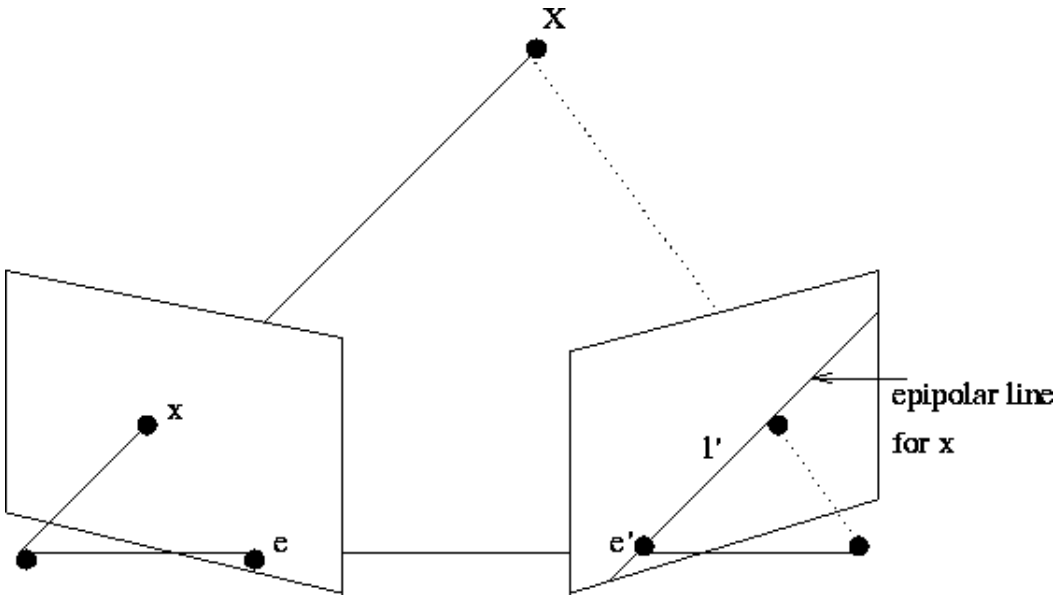
For a projection matrix $\mathbf{P}$, we could easily do a back-projection by calculating the pseudo inverse matrix $\mathbf{P}^+ = \mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1}$.
Therefore,we get that for a 2D point $\mathbf{x}$, it's back-projection $P^+\mathbf{x} = C + \lambda v$ (assuming the dot product $v.m^3 = 1$) resides in a plane parallel to the image screen, and that plane is of distance λ from the camera center.
Resulting in the following calculation for back-projection:

$$v_x = \frac{P^+\mathbf{x} - C}{dot(P^+\mathbf{x} - C, m^3)} \qquad X(\lambda) = \lambda v_x + C \tag{2}$$

where v_x is normalized such that $v.m^3 = 1$, $X(\lambda)$ is the back-projection of $\mathbf{x}$
to distance λ from the camera, and C is the point of the camera center as calculated in 1.

Back-projection , Triangulation and the Epipolar Line

Given a depth z, we use the the camera matrix to perform a back-projection from a 2D point on the screen, into a 3D point of distance z from the principle plane.
For each filmed pixel x, we calculated the possible locations of the filmed pixel.
Extending a line from the camera center through that pixel to infinity would represent the possible location in the 3D world.
Moreover, projecting that line into the projector screen gives us the epipolar line.



The epipolar line gives us the possible colors being projected from the projector into the world and observed by our camera in pixel x.
given two elements of these: x , a point on the epipolar line , z - we can "complete a triangle" and find the third element.
this process is called triangulation.

4 Reconstruction Model[1]

4.1 The Probability Model - Mathematical background

Using normal distribution, we have the following models:

regular case for calculating probability

$p(n|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(n-\mu)^2}{2\sigma^2}}$

Given the model we can calculate the probability of getting different data sets.

inverse probability

However, what we would like to know is the value of $p(\mu,\sigma^2|n)$

so given the measured data set we attempted to calculate the probability for different models.
using the following known identities:

$$p(a,b) = p(a)p(b|a) = p(b)p(a|b)$$

we get:

$$p(\mu,\sigma^2|n) = \frac{p(\mu,\sigma^2,n)}{p(n)} = \frac{p(\mu,\sigma^2)}{p(n)} p(n|\mu,\sigma^2)$$

note that after a measurement has been made, $p(n)$ is constant.
$p(\mu,\sigma^2)$ is called the prior.
$p(n|\mu,\sigma^2)$ is the likelihood.

Maximum Likelihood Estimator

assuming that the true values of μ,σ^2 are fixed, we produce an **estimation**:

$$\mathcal{L}\left(\mu,\sigma^2|n\right) = k(n)p(n|\mu,\sigma^2) \propto p(\mu,\sigma^2|n)$$

where:

- $\mathcal{L}\left(\mu,\sigma^2|n\right)$ is the likelihood.
- $k(n) = \frac{p(\mu,\sigma^2)}{p(n)}$, assumed to be a function of the data.

Since $p(n|\mu,\sigma^2)$ can be calculated, we can now compare likelihoods to
find μ,σ^2 that maximize $p(\mu,\sigma^2|n)$.
NOTE: we can only compare likelihoods for the same data-set due
to possibly changing constant factor.

4.2 Probabilistic Formulation

In order to extract the object's depth, we formulate a probability function,
then search for the the depth , that maximizes that function;
given the projected patterns and the taken pictures.

$$z = \underset{z,a,b}{argmax} P(z, a, b | I_C, I_P) \tag{3}$$

reformulating and removing constant factors gives us:

$$P(z, a, b | I_C, I_P) = \frac{P(z, a, b, I_C, I_P)}{P(I_C, I_P)} = \frac{P(a, b, I_C, I_P | z) P(z)}{P(I_C, I_P)} \propto P(a, b, I_C, I_P | z) P(z)$$

using the above result we reformulate:

$$z = \underset{z,a,b}{argmax} P(z, a, b | I_C, I_P) = \underset{z,a,b}{argmax} \left[P(a, b, I_C, I_P | z) P(z) \right]$$

then taking the negative log probability:

$$z = \underset{z,a,b}{argmin} \left[-logP(a, b, I_C, I_P | z) - logP(z) \right] \tag{4}$$

where:

- $P(a, b, I_C, I_P | z)$ is the photo-consistency term.
- $P(z)$ is the object's surface shape prior.

4.3 Photo-consistency Assumption

We assume a lambertian model with added Gaussian noise.
Thus, we describe the color seen at pixel x as follows:

$$I_{CAMERA}(x) = a(x) \cdot I_{PROJECTOR}(\Pi(x, z)) + b(x) + n(x) \tag{5}$$

- $a(x)$ and $b(x)$ depend on the object properties (albedo, color, surface angle etc).
- $\Pi(x, z)$ describes the corresponding pixel in the projector pattern (on the epipolar line) being projected on the object, assuming the object is at distance z from the camera center.
- $I_{PROJECTOR}(\Pi(x, z))$ describes the color being projected.
- $n(x)$ describes a Gaussian noise at a given pixel $n(x) \sim N(0, \sigma^2)$, assumed to originate from the camera sensor and is independent of neighboring pixels.
- given z and our photo-consistency assumption, we see a conditional independence of neighboring pixels values $I_{CAMERA}(x),\ I_{PROJECTOR}(\Pi(x, z)),\ a(x),\ b(x),\ n(x)$.

4.4 Optimization Scheme

Reviewing our photo-consistency assumption, the conditional independence of neighboring pixels values of $I_C(x),\ I_P(\Pi(x,z))\ a(x),\ b(x)$ gives us:

$z = \underset{z,a,b}{argmin}\left[\int_x \left(-logP(a,b,I_C,I_P|z)\right)dx - logP(z)\right]$

now assuming the optimal a, b are a function of z, I_C, I_P:

$z = \underset{z}{argmin}\left[\int_x \min_{a,b}\left(-logP(a,b,I_C,I_P|z)\right)dx - logP(z)\right] \Longrightarrow$

$$\mathbf{z \approx \underset{z}{argmin}\left[\rho_{SL}(z) + \psi(z)\right]} \tag{6}$$

- $\rho_{SL}(z)-$ is our penalty term for the photo-consistency assumption, optimized per pixel.
- $\psi(z)-$ is the penalty term for the object's surface shape, estimating the negative log probability prior.

In the following sections, we describe how to calculate such z that will minimize each penalty term separately.
Then describe an alternating minimization method that takes into account both penalty terms.

4.5 Calculation of Photo-consistency Term - $\rho_{SL}(z)$

As mentioned before, this term is optimized per pixel.
Therefore, in this section we mark $I_C(x),\ I_P(\Pi(x,z)),\ a(x),\ b(x)$ as $I_C,\ I_P,\ a,\ b$ respectively.

Based on the probability theory, we know that $P(a,b,I_C,I_P|z) = P(I_C,I_P|a,b,z)P(a,b|z)$.
we then incorporate a maximum likelihood estimation for a, b :

$$P(a,b,I_C,I_P|z) \propto P(I_C,I_P|a,b,z)$$

remembering that our photo-consistency term :

$$n = I_C - a\cdot I_P - b$$

we get:

$$P(I_C,I_P|a,b,z) = P(n|n = I_C - a\cdot I_P - b) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{n^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(I_C - a\cdot I_P - b)^2}{2\sigma^2}}$$

applying the log on the above probability we get:

$$log\ P(I_C,I_P|a,b,z) = -\frac{1}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\left(a\cdot I_P + b - I_C\right)^2$$

maximizing $-\frac{1}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\left(a\cdot I_P + b - I_C\right)^2 \Leftrightarrow$ minimizing $\left(a\cdot I_P + b - I_C\right)^2$
therefore over N conditionally independent patterns we get:

$$log\ \prod_{i=1}^{N} P(I_C[i],I_P[i]|a,b,z) = \sum_{i=1}^{N} log\ P(I_C[i],I_P[i]|a,b,z)$$

overall we get that the optimal choice for a, b minimize:

$$\sum_{i=1}^{N}\left(a\cdot I_P[i] + b - I_C[i]\right)^2$$

which can be optimized using the least squares method.

more generally we formulate this as an optimization problem: $f(a,b) = \sum_{i=1}^{N}\left(a\cdot I_P[i] + b - I_C[i]\right)^2$

the condition for stationary points is: $\frac{\partial f}{\partial a} = 0$ and $\frac{\partial f}{\partial b} = 0$.

$$\frac{\partial f}{\partial a} = \sum_{i=1}^{N} 2\left(a \cdot I_P[i] + b - I_C[i]\right) \cdot I_P[i] = 0 \;\Leftrightarrow\; a\left(\sum_{i=1}^{N} I_P[i]^2\right) + b\left(\sum_{i=1}^{N} I_P[i]\right) = \left(\sum_{i=1}^{N} I_C[i] \cdot I_P[i]\right)$$

$$\frac{\partial f}{\partial b} = \sum_{i=1}^{N} 2\left(a \cdot I_P[i] + b - I_C[i]\right) = 0 \;\Leftrightarrow\; a\left(\sum_{i=1}^{N} I_P[i]\right) + b\left(\sum_{i=1}^{N} 1\right) = \left(\sum_{i=1}^{N} I_C[i]\right)$$

giving us the following matrix representation to the problem:

$$\begin{array}{ll} \mu_{PP} \triangleq \sum_{i=1}^{N} I_P[i]^2 & \mu_P \triangleq \sum_{i=1}^{N} I_P[i] \\ \mu_{CP} \triangleq \sum_{i=1}^{N} I_C[i] \cdot I_P[i] & \mu_C \triangleq \sum_{i=1}^{N} I_C[i] \end{array} \Longrightarrow \begin{pmatrix} \mu_{PP} & \mu_P \\ \mu_P & N \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \mu_{CP} \\ \mu_C \end{pmatrix}$$

ignoring numerical stability issues, the most simple solution is given by:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \mu_{PP} & \mu_P \\ \mu_P & N \end{pmatrix}^{-1} \begin{pmatrix} \mu_{CP} \\ \mu_C \end{pmatrix} \tag{7}$$

giving us the calculation of the photo-consistency penalty term, calculated per pixel:

$$\rho_{\mathbf{SL}}(\mathbf{z}; \mathbf{I_C}, \mathbf{I_P}, \mathbf{x}) = \sum_{\mathbf{i=1}}^{\mathbf{N}} \left(\mathbf{a} \cdot \mathbf{I_P}[\mathbf{i}] + \mathbf{b} - \mathbf{I_C}[\mathbf{i}]\right)^{\mathbf{2}} \tag{8}$$

5 Prior Selection

Our assumption for the surface prior is piece-wise smoothness, we incorporate this assumption by minimizing the total variation of the surface.

Mathematical Introduction

Calculus of Variations and the Euler Lagrange Equations

Calculus of variations deals with optimizations of functionals.

A functional is a map from a vector space to A scalar field.

For example, suppose our vector space is the space of two variable functions,
and our scalar field is $\mathbb{R}$, then for function $u(x, y)$,
a possible functional f , would be $f[u] = \int_0^1\int_0^1 |u(x, y)|dxdy$

Minimizing a functional means finding the function $u(x, y)$ that minimizes $f[u]$.
in this case the solution is trivial,$f[u]$ is minimal for $u(x, y) = 0$.

NOTICE: calculus of variations deals with functionals of functions and not of discrete vectors;
although in practice we translate the results to discrete vectors (images etc).
For example:
translating the last functional into discrete math we get $u \in \mathbb{R}^{n\times m},\ f[u] = \sum_{i=1}^{n}\sum_{j=1}^{m} |u(i, j)|$

Euler Lagrange Equations For a certain set of functionals, the Euler Lagrange equations provide a necessary condition for optimality.

more specifically, for functionals of the form:

$$f[u] = \int_a^b L(x, u(x), u'(x))dx$$

where $L(x, u(x), u'(x))$ is some function with continuous first partial derivatives.
then the Euler Lagrange equation is:

$$\frac{\partial L}{\partial u} - \frac{d}{dx}\frac{\partial L}{\partial u'} = 0$$

and optimal solutions satisfy this equations.

Extension for two variables For our purposes, the following formulation is most useful:

$$f[u] = \int_a^b\int_c^d L(x, y, u(x, y), u_x(x, y), u_y(x, y))dxdy$$

then an optimal function $u(x, y)$, satisfies:

$$\frac{\partial L}{\partial u} - \frac{\partial}{\partial x}\frac{\partial L}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial L}{\partial u_y} = 0$$

Operators, Fourier Transform and Convolution Theorem

A **Linear Operator** $\hat{A}$, is an operator that for any two functions f, g

and any two scalars α, β holds that: $\hat{A}(\alpha f + \beta g) = \alpha \hat{A} f + \beta \hat{A} g$.

Example: $\frac{d}{dt} \Rightarrow \frac{d}{dt}(\alpha f + \beta g) = \alpha \frac{d}{dt} f + \beta \frac{d}{dt} g$

A **Time Invariant Operator** $\hat{A}$, is an operator that given $\hat{A}\{x(t)\} = y(t)$,

$\hat{A}$ holds that: $\hat{A}\{x(t+d)\} = y(t+d)$.

Example: $\frac{d}{dt} \Rightarrow \frac{d}{dt} x(t+d) = x'(t+d)$

Convolution is a specific kind of integral product between two functions, marked $f * g$.
If $\hat{A}$ is a **linear** and **time invariant** operator, there exists a function a (called kernel), such that for any function f we get $\hat{A} f = a * f$.
Convolution can be thought of as a generalization of matrix multiplication for the case where our vectors are non discrete (continuous functions).

Fourier Transform is a specific kind of integral product, different from convolution.
The Fourier transform of function g is marked $\mathcal{F}\{g\}$.
it is reversible $\mathcal{F}^{-1}\{\mathcal{F}\{g\}\} = g$, and linear.

Convolution Theorem
given a convolution $f * g$, we get under Fourier transform that $\mathcal{F}\{f * g\} = \mathcal{F}\{f\}\mathcal{F}\{g\}$.
This allows easy manipulation of equations with operators expressed as convolutions.

5.1 Total Variation Minimization

we shall use the following model:

$$f = Ku + n$$

where u is the original image, Ku is the distorted image, n is added noise.
f is the image acquired, we wish to extract u from f.
in our implementation we assume $K = I$.

our smoothing model is based on the following simple rule, find u_{TV}:

$$u_{TV} = \underset{u \in V}{argmin}\{||\nabla u|| + \frac{\alpha}{2}||Ku - f||^2\}$$

notice the different data types:

$$\nabla u_{i,j} = (\frac{\partial}{\partial x} u_{i,j}, \frac{\partial}{\partial y} u_{i,j}) \Rightarrow ||\nabla u_{i,j}|| = \sqrt{(\frac{\partial}{\partial x} u_{i,j})^2 + (\frac{\partial}{\partial y} u_{i,j})^2}$$

$$||\nabla u|| = \sum_{i,j} ||\nabla u_{i,j}||$$

basically we wish to minimize the absolute sum of the differentials (total variation).
let us look at the following expression:

$$\underbrace{||\nabla u||}_{Total\,Variation} + \underbrace{\frac{\alpha}{2}||Ku - f||^2}_{*}$$

* - represents the distance (difference) between the smoothed picture u and the originally acquired picture f.
since we don't wish the picture to be too smooth we will use the parameter $\frac{\alpha}{2}$ to give weight to this factor.
Optimization Method
the main "trick" here is going from a constrained single variable problem into unconstrained many variables and alternating optimization.
meaning we alternate the variables, and optimize it while treating the other variables as constants, hoping our solution will converge.

Variable Separation

set $p \approx \nabla u$ and get:

$$||\nabla u|| + \frac{\alpha}{2}||Ku - f||^2 \Rightarrow ||p|| + \frac{\alpha}{2}||Ku - f||^2$$

notice $p = 0$ gives a minimal expression so we must add a term that will ensure $p \approx \nabla u$

$$||p|| + \frac{\alpha}{2}||Ku - f||^2 \Rightarrow ||p|| + \frac{r}{2}||p - \nabla u||^2 + \frac{\alpha}{2}||Ku - f||^2$$

Just like $\frac{\alpha}{2}$, we have $\frac{r}{2}$ that will set how strongly we enforce this constraint.
changing this parameter will affect convergence and rate of convergence.
A high value will ensure convergence but will be slow.
A low value will advance faster but may not enforce constraint on p well enough, preventing convergence.

Using the augmented lagrangian

we have the penalty term $\frac{r}{2}||p - \nabla u||^2$ enforcing the constraint $p \approx \nabla u$,

we can also use the augmented lagrangian method by adding the term $\langle \lambda, p - \nabla u \rangle$:

$$||p|| + \frac{r}{2}||p - \nabla u||^2 + \frac{\alpha}{2}||Ku - f||^2 \Rightarrow ||p|| + \frac{r}{2}||p - \nabla u||^2 + \frac{\alpha}{2}||Ku - f||^2 + \langle \lambda, p - \nabla u \rangle$$

along with the updates:

$$\lambda^k = \lambda^{k-1} + r\,(p - \nabla u) \tag{9}$$

Final Functional

combine it all together - our variables are - p, u, λ and our final functional is:

$$Target = ||p|| + \frac{r}{2}||p - \nabla u||^2 + \frac{\alpha}{2}||Ku - f||^2 + \underbrace{\langle \lambda, p \rangle - \langle \lambda, \nabla u \rangle}_{=\langle \lambda, p - \nabla u \rangle}$$

Using an alternating optimization process we get the following algorithm:

1. initiate $\lambda^0 = 0$,$p^0 = 0$
2. do $L1$ times
   (a) optimize u using 10
   (b) optimize p using 12
   (c) update λ using 9

Optimizing for u

taking for the functional the terms depending on u and reformulating:

$$\frac{r}{2}||p - \nabla u||^2 + \frac{\alpha}{2}||Ku - f||^2 - \langle \lambda, \nabla u \rangle$$

$$\frac{r}{2}(p - \nabla u)^T(p - \nabla u) + \frac{\alpha}{2}(Ku - f)^T(Ku - f) - \langle \lambda, \nabla u \rangle$$

$$\frac{r}{2}\left[p^T(p - \nabla u) - \nabla u^T(p - \nabla u)\right] + \frac{\alpha}{2}\left[u^T K^T(Ku - f) - f^T(Ku - f)\right] - \langle \lambda, \nabla u \rangle$$

$$\frac{r}{2}\left[p^T p - p^T \nabla u - \nabla u^T p + \nabla u^T \nabla u\right] + \frac{\alpha}{2}\left[u^T K^T K u - u^T K^T f - f^T K u + f^T f\right] - \langle \lambda, \nabla u \rangle$$

$$\frac{r}{2}\left[p^T p - p^T \nabla u - \nabla u^T p + \nabla u^T \nabla u\right] + \frac{\alpha}{2}\left[(Ku)^T K u - u^T K^T f - f^T K u + f^T f\right] - \lambda^T \nabla u$$

again taking only terms that depend on u:

$$\frac{r}{2}\left[-\underbrace{p^T \nabla u}_{scalar} - \underbrace{\nabla u^T p}_{scalar} + \nabla u^T \nabla u\right] + \frac{\alpha}{2}\left[(Ku)^T K u - \underbrace{u^T K^T f}_{scalar} - \underbrace{f^T K u}_{scalar}\right] - \lambda^T \nabla u$$

for terms that are scalars we get these identities:

$$p^T \nabla u = (p^T \nabla u)^T = \nabla u^T p \quad ; \quad u^T K^T f = (u^T K^T f)^T = f^T K u = (K^T f)^T u$$

using these identities to reformulate:

$$\frac{r}{2}\left[-2p^T \nabla u + \nabla u^T \nabla u\right] + \frac{\alpha}{2}\left[(Ku)^T K u - 2(K^T f)^T u\right] - \lambda^T \nabla u = \mathcal{J}$$

now formulating the Lagrangian, and writing the operators as convolutions:

$$\mathcal{J} = \int_{x,y} \mathcal{L}(x, y, u, u_x, u_y) dx dy =$$

$$= \int_{x,y} \frac{r}{2}\left[-2p^x u_x - 2p^y u_y + u_x^2 + u_y^2\right] + \frac{\alpha}{2}\left[(k * u)^2 - 2(k^T * f)u\right] - \left[\lambda^x u_x + \lambda^y u_y\right] dx dy$$

therefore the integrand is:

$$\Rightarrow \mathcal{L}(x, y, u, u_x, u_y) = \frac{r}{2}\left[-2p^x u_x - 2p^y u_y + u_x^2 + u_y^2\right] + \frac{\alpha}{2}\left[(k * u)^2 - 2(k^T * f)u\right] - \left[\lambda^x u_x + \lambda^y u_y\right]$$

using the Euler Lagrange equations for stationary point:

$$\frac{\partial \mathcal{L}}{\partial u} - \frac{\partial}{\partial x}\frac{\partial \mathcal{L}}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial \mathcal{L}}{\partial u_y} = 0$$

$$\frac{\partial \mathcal{L}}{\partial u} = \frac{\alpha}{2}\left[2 \cdot (k * u)(k * 1) - 2(k^T * f)\right] = \alpha\left[(k^2 * u) - (k^T * f)\right]$$

$$\frac{\partial \mathcal{L}}{\partial u_x} = r\left[-p^x + u_x\right] - \lambda^x \Rightarrow \frac{\partial}{\partial x}\frac{\partial \mathcal{L}}{\partial u_x} = r\left[-p^x_x + u_{xx}\right] - \lambda^x_x$$

$$\frac{\partial \mathcal{L}}{\partial u_y} = r\left[-p^y + u_y\right] - \lambda^y \Rightarrow \frac{\partial}{\partial y}\frac{\partial \mathcal{L}}{\partial u_y} = r\left[-p^y_y + u_{yy}\right] - \lambda^y_y$$

$$\implies \frac{\partial \mathcal{L}}{\partial u} - \frac{\partial}{\partial x}\frac{\partial \mathcal{L}}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial \mathcal{L}}{\partial u_y} = \alpha\left[(k^2 * u) - (k^T * f)\right] + r\left[p^x_x + p^y_y\right] - r\left[u_{xx} + u_{yy}\right] + \left[\lambda^x_x + \lambda^y_y\right] = 0$$

thus giving us a condition for optimal solution:

$$\implies \alpha K^T K u - \alpha K^T f + r \cdot div(p) - r\triangle u + div(\lambda) = 0$$

since all operators are linear and shift invariant, we can write them using convolution:

$$Au = (h_A * u)$$

additionally, using convolution theorem, we can use point wise operation instead of matrix multiplication:

$$\mathcal{F}\{Au\} = \mathcal{F}\{h_A * u\} = \mathcal{F}\{A\} \circ \mathcal{F}\{u\} \implies$$

$$\mathcal{F}\left\{\alpha K^T K u - \alpha K^T f + r \cdot div(p) - r\triangle u + div(\lambda)\right\} = 0$$

$$\mathcal{F}\left\{\alpha K^T K u\right\} - \mathcal{F}\left\{\alpha K^T f\right\} + \mathcal{F}\{r \cdot div(p)\} - \mathcal{F}\{r\triangle u\} + \mathcal{F}\{div(\lambda)\} = 0$$

$$\alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{K\} \circ \mathcal{F}\{u\} - \alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{f\} + r\mathcal{F}\{D_x p^x + D_y p^y\} - r\mathcal{F}\{\triangle\} \circ \mathcal{F}\{u\} + \mathcal{F}\{D_x \lambda^x + D_y \lambda^y\} = 0$$

$$\alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{K\} \circ \mathcal{F}\{u\} - \alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{f\} + r\mathcal{F}\{D_x\} \circ \mathcal{F}\{p^x\} + r\mathcal{F}\{D_y\} \circ \mathcal{F}\{p^y\} - r\mathcal{F}\{\triangle\} \circ \mathcal{F}\{u\} + \mathcal{F}\{D_x\} \circ \mathcal{F}\{\lambda^x\} + \mathcal{F}\{D_y\} \circ \mathcal{F}\{\lambda^y\} = 0$$

reformulating the left term:

$$\alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{K\} \circ \mathcal{F}\{u\} - r\mathcal{F}\{\triangle\} \circ \mathcal{F}\{u\} - \alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{f\} + r\mathcal{F}\{D_x\} \circ \mathcal{F}\{p^x\} + \mathcal{F}\{D_x\} \circ \mathcal{F}\{\lambda^x\} + r\mathcal{F}\{D_y\} \circ \mathcal{F}\{p^y\} + \mathcal{F}\{D_y\} \circ \mathcal{F}\{\lambda^y\} = 0$$

taking out common factors:

$$\left[\alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{K\} - r\mathcal{F}\{\triangle\}\right] \circ \mathcal{F}\{u\} - \alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{f\} + \mathcal{F}\{D_x\} \circ [\mathcal{F}\{\lambda^x\} + r\mathcal{F}\{p^x\}] + \mathcal{F}\{D_y\} \circ [\mathcal{F}\{\lambda^y\} + r\mathcal{F}\{p^y\}] = 0$$

use linearity:

$$\left[\alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{K\} - r\mathcal{F}\{\triangle\}\right] \circ \mathcal{F}\{u\} - \alpha\mathcal{F}\left\{K^T\right\} \circ \mathcal{F}\{f\} + \mathcal{F}\{D_x\} \circ \mathcal{F}\{r \cdot p^x + \lambda^x\} + \mathcal{F}\{D_y\} \circ \mathcal{F}\{r \cdot p^y + \lambda^y\} = 0$$

rearrange terms:

$$\left[\alpha\mathcal{F}\left\{K^T\right\}\circ\mathcal{F}\{K\}-r\mathcal{F}\{\triangle\}\right]\circ\mathcal{F}\{u\}=\alpha\mathcal{F}\left\{K^T\right\}\circ\mathcal{F}\{f\}-\mathcal{F}\{D_x\}\circ\mathcal{F}\{r\cdot p^x+\lambda^x\}-\mathcal{F}\{D_y\}\circ\mathcal{F}\{r\cdot p^y+\lambda^y\}$$

divide:

$$\mathcal{F}\{u\}=\frac{\alpha\mathcal{F}\left\{K^T\right\}\circ\mathcal{F}\{f\}-\mathcal{F}\{D_x\}\circ\mathcal{F}\{r\cdot p^x+\lambda^x\}-\mathcal{F}\{D_y\}\circ\mathcal{F}\{r\cdot p^y+\lambda^y\}}{\alpha\mathcal{F}\{K^T\}\circ\mathcal{F}\{K\}-r\mathcal{F}\{\triangle\}}$$

and performing the inverse transform we get a closed formula for optimal solution:

$$u=\mathcal{F}^{-1}\left(\frac{\alpha\mathcal{F}\left\{K^T\right\}\circ\mathcal{F}\{f\}-\mathcal{F}\{D_x\}\circ\mathcal{F}\{r\cdot p^x+\lambda^x\}-\mathcal{F}\{D_y\}\circ\mathcal{F}\{r\cdot p^y+\lambda^y\}}{\alpha\mathcal{F}\{K^T\}\circ\mathcal{F}\{K\}-r\mathcal{F}\{\triangle\}}\right) \tag{10}$$

Optimizing for p

taking from the functional only terms dependent on p:

$$||p||+\frac{r}{2}||p-\nabla u||^2+\langle\lambda,p-\nabla u\rangle$$

now reformulating, we use variable substitution to solve an equivalent problem:
marking:

$$w(x)\equiv\nabla u(x)-\frac{\lambda}{r} \tag{11}$$

we get:

$$||p||+\frac{r}{2}||p-w||^2=$$

$$=\int_x||p(x)||+\frac{r}{2}\left[p(x)-w(x)\right]^2=\int_x||p(x)||+\frac{r}{2}\left[p(x)^2-2p(x)w(x)+w(x)^2\right]=$$

$$=\int_x||p(x)||+\frac{r}{2}\left[p(x)^2-2p(x)\left(\nabla u(x)-\frac{\lambda}{r}\right)+\left(\nabla u(x)-\frac{\lambda}{r}\right)^2\right]=$$

$$=\int_x||p(x)||+\frac{r}{2}\left[p(x)^2-2p(x)\nabla u(x)+2p(x)\frac{\lambda}{r}+\nabla u(x)^2-2\nabla u(x)\frac{\lambda}{r}\right]=$$

$$=\int_x||p(x)||+\frac{r}{2}\left[p(x)^2-2p(x)\nabla u(x)+\nabla u(x)^2\right]+\frac{r}{2}\left[2p(x)\frac{\lambda}{r}-2\nabla u(x)\frac{\lambda}{r}\right]=$$

$$=\int_x||p(x)||+\frac{r}{2}\left[p(x)^2-2p(x)\nabla u(x)+\nabla u(x)^2\right]+\lambda p(x)-\lambda\nabla u(x)=$$

$$=||p||+\frac{r}{2}||p-\nabla u||^2+\langle\lambda,p-\nabla u\rangle$$

meaning that minimizing:

$$w=\nabla u-\frac{\lambda}{r}\quad;\ ||p||+\frac{r}{2}||p-w||^2$$

will also minimize the original functional:

$$||p||+\frac{r}{2}||p-\nabla u||^2+\langle\lambda,p-\nabla u\rangle$$

by close observation of the term:

$$||p|| + \frac{r}{2}||p - w||^2 = \int_x ||p(x)|| + \frac{r}{2}\left(p(x) - w(x)\right)^2$$

we notice we can minimize the functional by minimizing each component separately, the per point condition is:

$$p(x) = \begin{cases} \left(1 - \frac{1}{r}\frac{1}{||w(x)||}\right) w(x) & ||w(x)|| > \frac{1}{r} \\ 0 & else \end{cases} \tag{12}$$

6 Alternating Photo-consistency Term with Total Variation Term

recalling our overall optimization scheme, our functional is

$\mathbf{z} \approx \underset{\mathbf{z}}{\mathbf{argmin}} \left[\rho_{\mathbf{SL}}(\mathbf{z}) + ||\nabla \mathbf{z}||\right]$

Setting $u = z$, we turn this into an alternating minimization

$\mathbf{z} \approx \underset{\mathbf{z}}{\mathbf{argmin}} \left[\rho_{\mathbf{SL}}(\mathbf{z}) + \frac{\alpha}{\mathbf{2}}||\mathbf{Ku} - \mathbf{z}||^{\mathbf{2}} + ||\nabla \mathbf{u}||\right]$

using the augmented lagrangian method, add the term for lagrangian multiplier term

$$z \approx \underset{z}{argmin} \left[\rho_{SL}(z) + \frac{\alpha}{2}||Ku - z||^2 + ||\nabla u|| + \langle \mu, Ku - z\rangle\right] \tag{13}$$

along with updates:

$$\mu = \mu + \alpha\left(z - u\right) \tag{14}$$

resulting in the updated optimization algorithm:

1. initiate $\lambda^0 = 0$,$p^0 = 0$,$\mu^0 = 0$
2. per pixel x
   (a) pick $z(x)$ that minimizes 8
3. do i times
   (a) do $L1$ times
      i. optimize u using 15
      ii. optimize p using 12
      iii. update λ using 9
   (b) update μ using 14
   (c) update z using 18

6.1 Update to Total Variation Minimization

our updated functional to minimize becomes:

$$||\nabla u|| + \frac{\alpha}{2}||Ku - z||^2 \Rightarrow ||\nabla u|| + \frac{\alpha}{2}||Ku - z||^2 + \langle \mu, Ku - z\rangle$$

further development in the same manner as before, gives us :

$$\frac{r}{2}\left[-2p^T\nabla u+\nabla u^T\nabla u\right]+\frac{\alpha}{2}\left[(Ku)^TKu-2(K^Tz)^Tu\right]-\lambda^T\nabla u+\underbrace{\mu^TKu}_{=(K^T\mu)^Tu}$$

it easy to verify that the appropriate Lagrangian term is:

$$\mathcal{L}(x,y,u,u_x,u_y)=\frac{r}{2}\left[-2p^xu_x-2p^yu_y+u_x^2+u_y^2\right]+\frac{\alpha}{2}\left[(k*u)^2-2(k^T*z)u\right]-[\lambda^xu_x+\lambda^yu_y]+(k^T*\mu)u$$

so the condition for a stationary point is:

$$\alpha K^TKu-\alpha K^Tz+r\cdot div(p)-r\triangle u+div(\lambda)+K^T\mu=0$$

reformulating to solve using Fourier transform we get:

$$u=\mathcal{F}^{-1}\left(\frac{\alpha\mathcal{F}\left\{K^T\right\}\circ\mathcal{F}\left\{z\right\}-\mathcal{F}\left\{K^T\right\}\circ\mathcal{F}\left\{\mu\right\}-\mathcal{F}\left\{D_x\right\}\circ\mathcal{F}\left\{r\cdot p^x+\lambda^x\right\}-\mathcal{F}\left\{D_y\right\}\circ\mathcal{F}\left\{r\cdot p^y+\lambda^y\right\}}{\alpha\mathcal{F}\left\{K^T\right\}\circ\mathcal{F}\left\{K\right\}-r\mathcal{F}\left\{\triangle\right\}}\right) \tag{15}$$

6.2 Update To Photo-Consistency Term

our functional is: $z\approx\underset{z}{argmin}\left[\rho_{SL}(z)+\frac{\alpha}{2}||Ku-z||^2+||\nabla u||+\langle\mu,Ku-z\rangle\right]$

taking only terms that depend on z, we get: $z\approx\underset{z}{argmin}\left[\rho_{SL}(z)+\frac{\alpha}{2}||Ku-z||^2-\langle\mu,z\rangle\right]$.

Optimizing this term requires re-sweeping of depth values, which is very costly.

However, thanks to the term $\frac{\alpha}{2}||Ku-z||^2$, we can limit our sweeping range after acquiring u.

We assume our original photo-consistency term, $\rho_{SL}(z;I_C,I_P,x)$, is piece wise linear within small intervals,

Therefore, given z^i from previous iterations we calculate the new photo-consistency term:

$$\tilde{\rho}_{ALT}(z)=\begin{cases}m_1z+b_1+\frac{\alpha}{2}\left(u-z\right)^2-\mu z & if\ z<z^i\\ m_2z+b_2+\frac{\alpha}{2}\left(u-z\right)^2-\mu z & else\end{cases} \tag{16}$$

where:

$$m_1=\frac{\rho_{SL}(z^i)-\rho_{SL}(z^i-\epsilon)}{\epsilon}\qquad b_1=(1-m_1)\rho_{SL}(z^i)$$

$$m_2=\frac{\rho_{SL}(z^i+\epsilon)-\rho_{SL}(z^i)}{\epsilon}\qquad b_2=(1-m_2)\rho_{SL}(z^i)$$

NOTICE:

- m_1,m_2,b_1,b_2 are acquired by calculating ρ_{SL} for only 3 discrete values, and with small constant value ϵ.
- our term for optimal z is now : $z^{i+1}=\underset{z}{argmin}\left[\tilde{\rho}_{ALT}(z)\right]$
- We only calculate $\tilde{\rho}_{ALT}(z)$ for two z values, due to each segment having it's own optimal z, calculated analytically.

Optimal depth per segment

$$\tilde{\rho}_{ALT}(z) = m_s z + b_s + \frac{\alpha}{2}(u - z)^2 - \mu z$$

this is a simple rank 2 polynom with a single minimum point,

we optimize it by differentiating and comparing to 0:

$$m_s - \alpha u + \alpha z - \mu = 0 \quad \Rightarrow \quad z_s^{opt} = u + \frac{\mu - m_s}{\alpha}$$

then we truncate the optimal value back into the segment

$$z_s^{trunc} = \begin{cases} min(z_1^{opt}, z^i) & s = 1 \\ max(z_2^{opt}, z^i) & s = 2 \end{cases} \tag{17}$$

Then pick the one that minimizes our term:

$$z^{i+1} = \underset{z \in \{z_1^{trunc}, z_2^{trunc}\}}{argmin} [\tilde{\rho}_{ALT}(z)] \tag{18}$$

7 Technical Details

7.1 Dynamic Sweep Approach

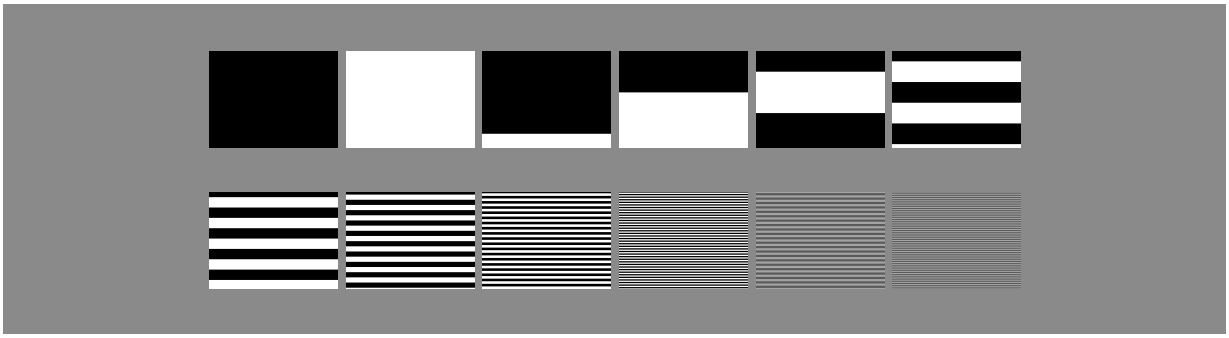
Since the photo-consistency is non-convex, non-differentiable, and may have multiple local minima. We are forced to sweep many depth values in order to find an optimal solution.
Assuming that the background of the object in the scene is static, we can reduce the required scans by sweeping more cleverly.
We offer doing a sparse global scan along with a dense local scan, where local is in the certain area of previously assumed depth.
The local scan will try to catch moving objects assuming they don't move too fast.
The sparse scan will complete a dense global scan every several frames, ensuring every depth is sweeped, and that we eventually find the right depth.
the global scan will hopefully get an estimated result that is near the real result, that will be catched in the following frame's local scan.

7.2 Projected Pattern[5][6]

Selecting a proper pattern has great importance to the robustness and correctness of the reconstruction. The algorithm must easily find for each filmed pixel, what is the corresponding pixel in the projected pattern being projected on the object's surface.
Different patterns may have different tradeoffs.

We used A Grey coded binary pattern that has the following properties:

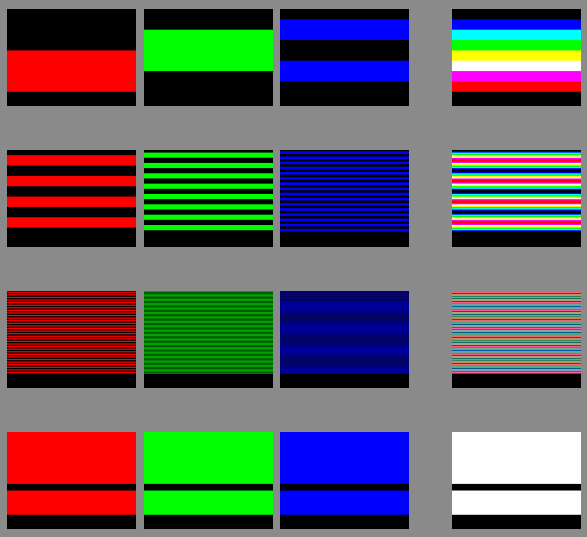
- the delta between the colors being projected is maximal, making it easier to detect what color was projected.
- the stripe pattern can be saved using a 1D array, saving memory accesses, can also be generated procedurally costing no memory.



it is worth noting that the algorithm was constructed and implemented for the general case, assuming the pattern is a generic 2D gray-scale image.

7.2.1 Color Separation[6]

To improve projection speed, several patterns were projected simultaneously in different channels (Red,Green,Blue),
then captured with an RGB camera, and separated back into the different channels.
This resulted in 3 times speedup.



on the left:Grey coded patterns in alternating channels

on the right: merged 3 channel picture being projected

Given that the object's properties (albedo) differ with color, we now require a modification of the algorithm,Mainly in photo-consistency assumption.
Therefore, when calculating the photo-consistency term, we need to find a, b separately per color channel.
However, separating the samples to different color channels means we are using only a third of the available data.

$$S_{color} = \{i | I_P[i] \; is \; of \; color \; c\}$$

We can improve this by making the following distinction:
Albedo and lighting properties depend on the actual color being projected.
Therefore, samples depicting the object's dark areas can be used for all 3 color channels:

$$S_{universal} = \{i | I_P[i] \leq threshold \; for \; "dark"\}$$

Allowing us to increase the number of samples used per channel:

$$S_{color}^{boost} = S_{color} \cup S_{universal}$$

We now search a^c, b^c separately per channel $c \in \{R, G, B\}$ that minimize:

$$(a^c, b^c) = \underset{a^c, b^c}{argmin} \left[\sum_{i \in S_c^{boost}} \left(a^c \cdot I_P[i] + b^c - I_C[i] \right)^2 \right] \tag{19}$$

after finding these terms per color channel, we now have a penalty term for photo-consistency per channel:

$$\rho_{SL}^c(z; I_C, I_P, x) = \sum_{i \in S_c} \left(a^c \cdot I_P[i] + b^c - I_C[i] \right)^2 \tag{20}$$

We assume noise levels per color channel are independent of each other.
Therefore, since our penalty terms estimate the negative log probability, we can get a combined penalty term:

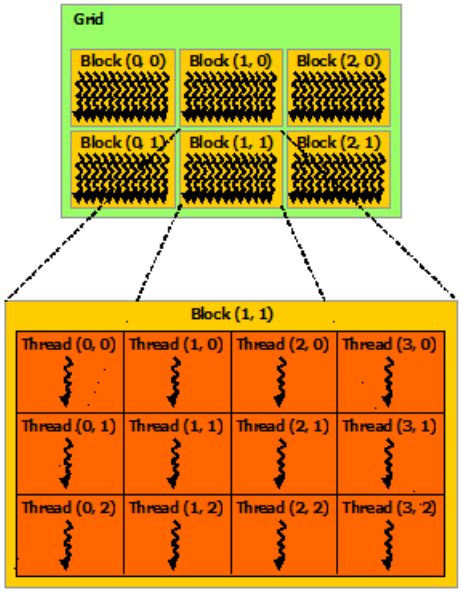
$$\rho_{SL}(z; I_C, I_P, x) = \sum_{c \in \{R,G,B\}} \rho_{SL}^c(z; I_C, I_P, x) \tag{21}$$

7.3 CUDA [9]

7.3.1 Cuda Computation Model

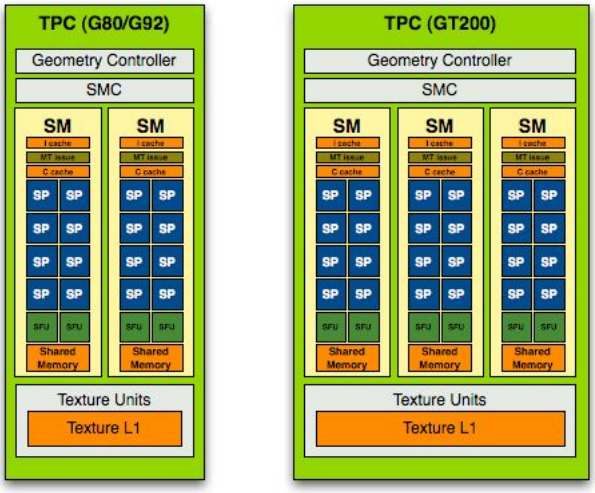
the Cuda computation model has two main hierarchies.

1. Grid of blocks - synonymous to a grid of CPUs.
2. Block of threads - synonymous to a CPU with several cores, each core is a thread.



7.3.2 Cuda Hardware Model

To put the computation model into context we shall describe how this model then translates into the metal.
the hardware has two main hierarchies.
1. Stream Multiprocessor - the SM dispatches instructions similar to a SIMD unit.
2. Stream Processor - the SP executes instructions sent from the SM on it's own copy of the data.
NOTES: different scaling factors.
The number of SM per card may differ.
The number of SPs per SM may differ.
The latency (number of clock cycles) per instruction may differ.



7.3.3 Instruction Scheduling

scheduling is what connects the grids and blocks to the actual hardware.

Warp each thread block is being divided into "warps" (batches of 32 threads)
Ideally all threads in a warp agrees on the executed instruction and execute in parallel.

Half-Warp\Quarter-Warp When accessing memory, request are being split into separate 128-byte memory requests.
So when a warp tries to access more than that, warps are divided into half and quarter warps then memory requests are issued serially per half\quarter warp.

Divergence in case a program path diverges, serial execution is performed.
therefore one should take care to split separate execution paths into separate warps.

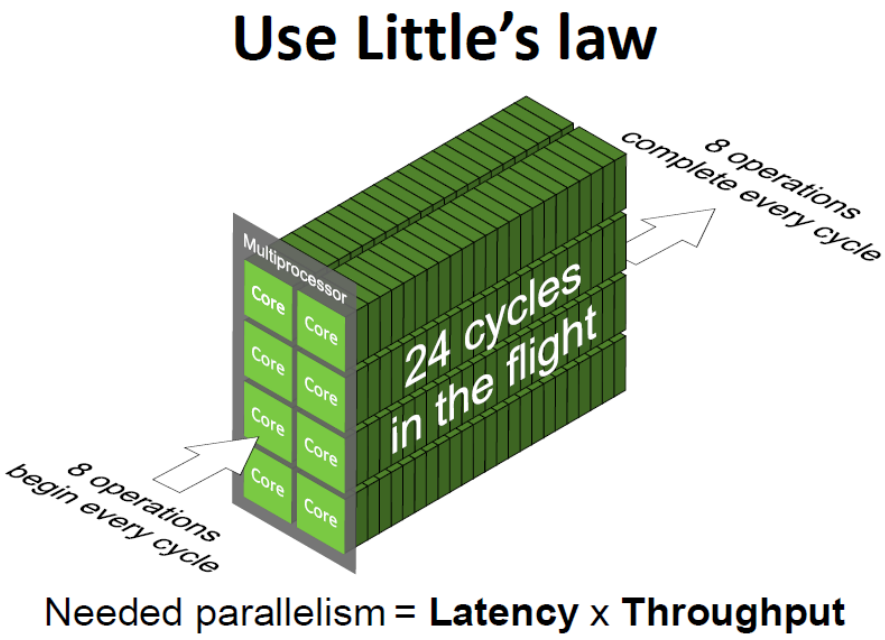
Execution Context each SM can be given one or more thread blocks to be executed on it,
each thread has it's own execution context that must be kept
throughout the thread's life time - registers, address space ,program counter etc.
therefore a given SM has a maximum number of threads it can track.

Occupancy - Thread Level Parallelism A way to hide latency by maximizing the number of threads in a block.
Blocks are divided into warps, more threads mean more warps can be pipe-lined, hiding instruction latency.

$$Occupancy = \frac{execution\ contexts\ active}{maximum\ possible\ execution\ contexts}$$

basically the number of thread running on a SM divided by the maximum possible.
this is determined by the amount of resources the threads takes.
It is not the only way to hide instruction latency,
therefore higher occupancy does not automatically mean better performance.

Warp Scheduler - Instruction Level Parallelism as the name suggest this unit dispatches instructions from executed warps to the SPs.
may issue one or more warp instruction per cycle (Instruction Level Parallelism).
broadcast that instruction to several SPs.
on some architectures must utilize ILP to get full utilization.



7.3.4 Computation Limitations

we shall try and estimate according to the algorithm the required computation power.
$W = picture\ width$ $H = picture\ height$
$PL = number\ of\ planes\ sweeped$
$FPS = desired\ reconstructed\ frames\ per\ second$
$AP = algorithm\ operations = W \times H \times FPS \times PL$
$N = number\ of\ patterns\ per\ reconstruction\ frame$

8 Cuda Optimizations Used

8.1 Thread Data Model

in our implementation for the plane sweeping we experimented with two primary thread-data model.
our data has two levels of calculation independency

1. each pixel's calculation is independent of other pixels.
2. each plane sweeped, is independent of other planes.

Our first implementation assigned a block per pixel, then a thread per plane.
Our second model assigned a pixel per thread, which then sweeped several planes.
The first model turned out to be less then optimal due to several reasons:

- Since each thread has a calculated probability for it's depth, we need to pick the best one over all the threads, adding an additional computation step, this step is not required in the second model.
- Since the number of threads per block is limited, this imposes a hard restriction on number of planes sweeped, while the second model has no restrictions.

- The memory access patterns of the first model had less spatial locality within a warp.

8.2 First Steps - Compute\Memory Bound Kernels

Before approaching a code optimization task, we must find the run-time bottle-necks.

Kernels are generally divided into memory bound kernels or compute bound kernels.

Trick for Checking if A kernel is Compute\Memory bound

Starting with compute capability 2.0 , the floating point operations conform to IEEE754-2008
as opposed to compute capability 1.x which conforms to faster, less accurate IEEE754-1985.
for devices capable of compute capability 2.0 and above, it is possible to switch between
the two standards using the "–use_fast_math" flag.
Considerable speed difference means that your kernel is likely compute bound.
Little to no speed difference means you kernel is likely memory bound.
It is also worth experimenting with different L1 cache sizes:
$cudaDeviceSetCacheConfig(cudaFuncCachePreferL1)$ - set larger L1 cache.
$cudaDeviceSetCacheConfig(cudaFuncCachePreferShared)$ - set smaller L1 cache.

8.3 Optimizing memory bound kernels

Memory bound kernels spend a lot of time waiting for data to be transferred from\to memory before continuing with the calculation.

This can be caused by a high rate of cache misses.

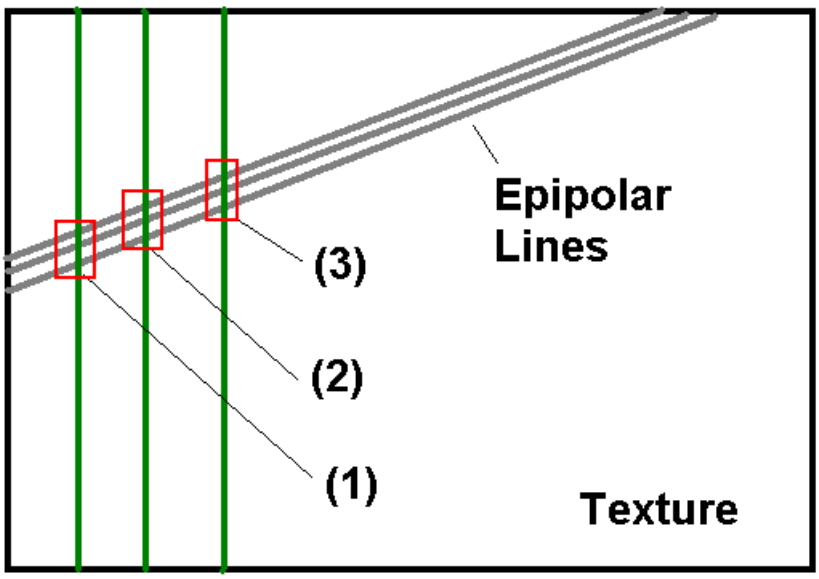
Or by low ratio between instructions executed to memory being accessed.

Most kernels are memory bound and usually resolving memory bottlenecks result in bigger performance gains.

There are several techniques for optimizing memory bottle-necks.

8.3.1 Texture Memory

Texture memory differs from regular memory by it's organization of the cache lines.
While regular global memory has a linearly ordered cache line, texture memory
has a spatially ordered cache line.
This is well suited for our case, our access pattern has a high degree of spatial
locality, since we linearly traverse the epipolar line on the projected pattern.
Moreover, nearby threads (threads within a warp) traverse close epipolar lines
further improving memory locality.



example of access pattern to the back-projection texture during a sweep.

numbers indicate access order.

In the above figure, we note several characteristics of our memory access pattern:
In grey, we see the epipolar lines originated from a group of nearby image pixels,
each assigned to a thread.
In green, we see a representation of the different planes being sweeped.
The chosen thread data model dictate that all threads within a warp sweep the
same plane in parallel.
Overall, each warp display a very high locality of memory access pattern (In red).
By making the plane sweep sampling denser, we get better locality, improving scalability.

8.3.2 Constant Memory

Constant memory, is a small, highly parallel read only memory, with a separate cache.
We used this memory to store common values being accessed by all threads simultaneously.
Primarily projection and back-projection matrices.
As long as all the threads access the same address, then access to constant memory is
even faster than shared memory due to constant memory not having bank conflicts.

8.3.3 Register Allocation and Local Memory

Register allocation can be a large memory bottle-neck.
Values not stored in registers are stored in local memory, which is essentially
a high latency global memory.
The number of registers available per thread is determined by the block size.
Each block has a set number of registers to be divided by all it's threads.
Managing register allocation can be tricky, with several trade-offs and
requires experimentation, as a rule of thumb:

- Avoid long and complex kernels that require many registers.
- Avoid accessing local arrays with run-time determined indexes, this automatically places the array in local memory.

- Consider recalculation of values rather than saving them for later use.
- Consider storing values for later use in shared memory.

Hints to register allocation issues can be obtained by looking at the PTX file generated by the compiler.
(using "–ptxas-options=-v -keep" compiler flags)
This outputs the maximum register usage per kernel.
Looking at the PTX file can give more specific insights.
NOTICE: PTX is only intermediate language, it goes through further optimizations during the compilation process, this output should not be mistaken for the actual final executable.

8.4 Optimizing compute bound kernels

Compute bound kernels continuously execute instructions.
This may include kernels with inefficient thread\instruction level parallelism, but also optimal kernels.
Optimizing compute bound kernels usually results in smaller performance gains and require more micro optimizations.
The best way to improve a compute bound kernel is to change the algorithm, which is not always an option.

8.4.1 Divergence Minimization

As explained before, divergent paths within a warp are executed serially.
This can have fatal effects on performance, and is hard to optimize where execution path depend on run-time data.
Such was the case in the calculation of 19.
This was especially problematic since this code lies in the heart of photo-consistency calculation and is considered a "Hot Spot".

```
217
218 for ( int k = 0 ; k<12 ; k++)
219 {
220
221     if( ( k%3 != color)  && (flags & g_flag_fast_fix  ) ) {
222         if(T_IP(k) >= threshold){
223             continue
224         }
225     }
226     float t_ip_k = T_IP(k);
227     float t_ic_k = T_IC(k);
228
229     sA(0) += t_ip_k*t_ip_k;
230     sA(1) += t_ip_k;
231     sA(2) += t_ip_k;
232     sA(3) += 1.0f;
233
234     B(0) += t_ic_k*t_ip_k;
235     B(1) += t_ic_k;
236 }
```

```
217
218 for ( int k = 0 ; k<12 ; k++)
219 {
220     float ttt=1.0f;
221     if( (MOD3(k) != color)  && (flags & g_flag_fast_fix  ) ) {
222         if(T_IP(k) >= threshold){
223             ttt=0.0f;
224         }
225     }
226     float t_ip_k = T_IP(k)*ttt;
227     float t_ic_k = T_IC(k)*ttt;
228
229     sA(0) += t_ip_k*t_ip_k;
230     sA(1) += t_ip_k;
231     sA(2) += t_ip_k;
232     sA(3) += ttt;
233
234     B(0) += t_ic_k*t_ip_k;
235     B(1) += t_ic_k;
236 }
```

per color calculation before matrix inversion seen at 7

left to right: unoptimized code vs optimized code.

Looking at the above code, we see a dynamic check for a threshold in line 222.
This results in a different execution paths taken by different threads.
For the unoptimized version this results in a long divergent path, spanning lines $222 - 236$.
We can shorten this path by using an extra variable (lines $220, 223$)
and modifying the calculation (lines $225, 226$).
As can be seen, the optimized version divergent path span a single operation (line 223).
Remarkably although more calculations are being made, the resulting extra parallelism increased performance.

8.4.2 Fast modulo operations

For certain types of calculations, we know the range of the input, allowing special optimizations. Such is the case for modulo operation, which basically requires an expensive division.
Common cases optimize the modulo operations for powers of 2, unlike our case requiring modulo by the number 3.
Continuing the above example, we know that the variable k is smaller that 16.
By writing the number as a sum of powers of 2, we get:

$$k \ mod \ 3 = (a \cdot 8 + b \cdot 4 + c \cdot 2 + d \cdot 1) \ mod \ 3 =$$

$$= (a \cdot (8 \ mod \ 3) + b \cdot (4 \ mod \ 3) + c \cdot (2 \ mod \ 3) + d \cdot (1 \ mod \ 3)) \ mod \ 3 =$$

$$= (a \cdot 2 + b \cdot 1 + c \cdot 2 + d \cdot 1) \ mod \ 3 = ((k \gg 2) + k \ \& \ 0x3) \ mod \ 3$$

looking just at $(k \gg 2) + k \ \& \ 0x3$ we know its maximal value is 6, giving us the following algorithm:

```
 9  inline __device__ int mod3( int a ) {
10      // works ok for numbers 0-15
11      a = (a >>  2) + (a & 0x3);
12      if (a > 2) a = a - 3;
13      return a;
14  }
```

9 Results

We compare our method using 3 structured light methods:

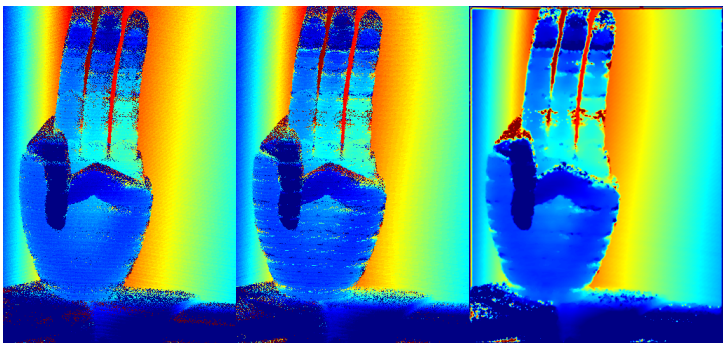
- method (a) classic reconstruction method, based on decoding and algebraic triangulation.
- method (b) our method without TV regularization.
- method (c) our method with TV regularization.

We also remind here the different calculation parameters in relation to TV regularization:

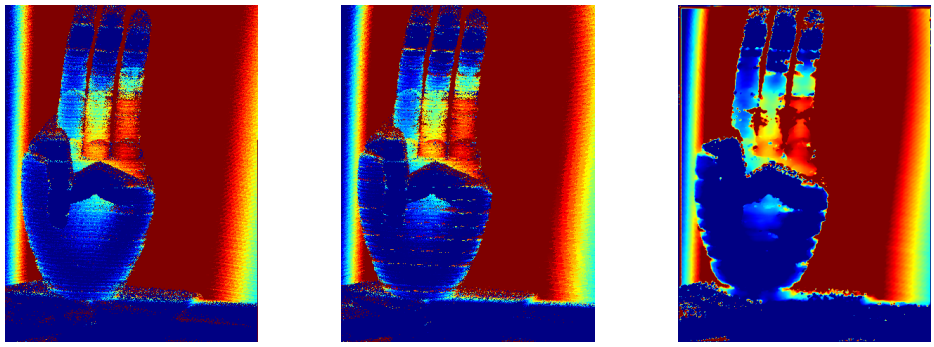
- i - number of alternating iterations
- $L1$ - number of inner TV iterations (Higher = Stronger Smooth)
- r - constraint strength for actually minimizing TV
- α - constraint strength for solution resembling original image.

We present reconstruction results with JET colored depth map.

9.1 Comparing with Existing techniques

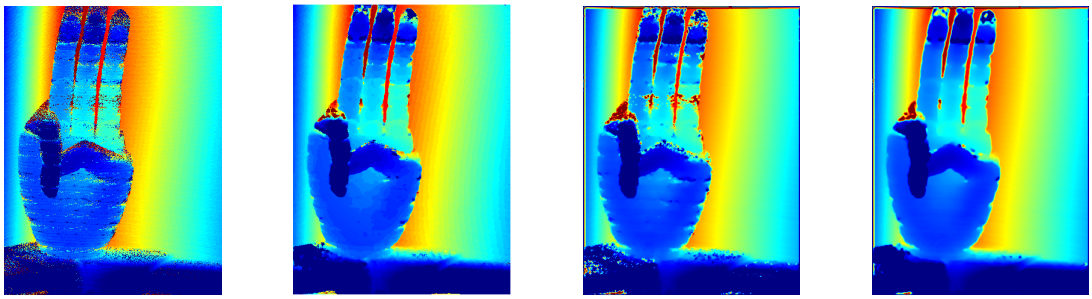


left to right: method (a); method (b); method (c)

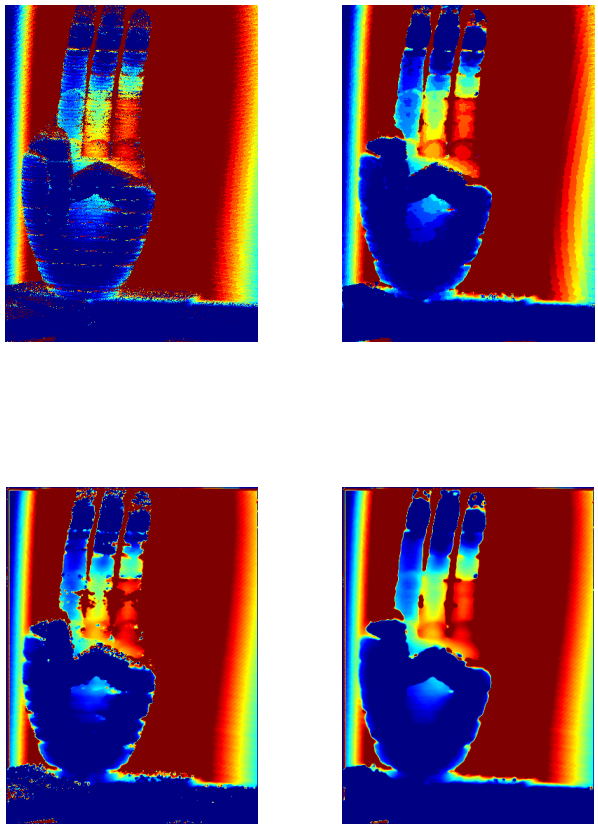


close up - left to right: method (a); method (b); method (c)

9.2 Comparing TV regularization with median filter



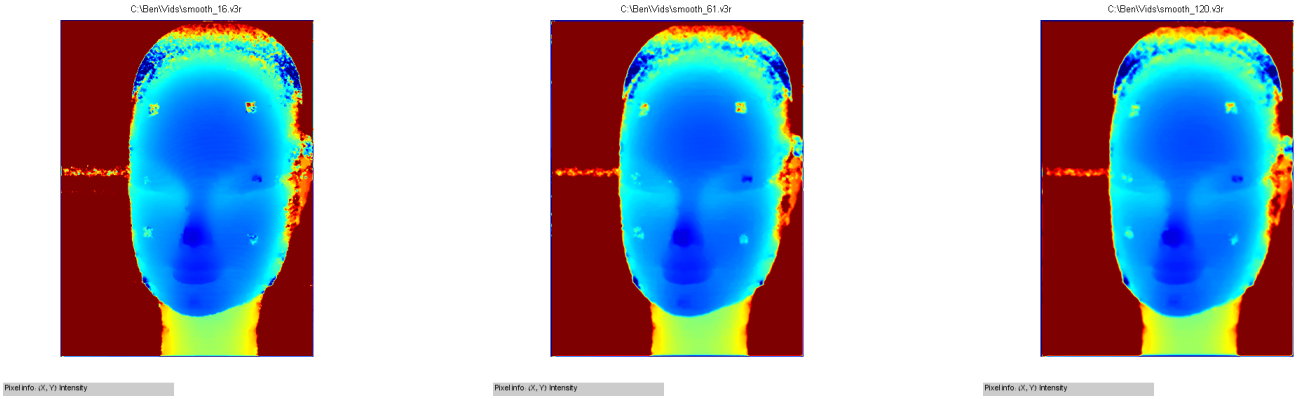
left to right: method (b); method (b) with 5x5 median filter; method (c); method (c) filtered with 5x5 median before TV regularization



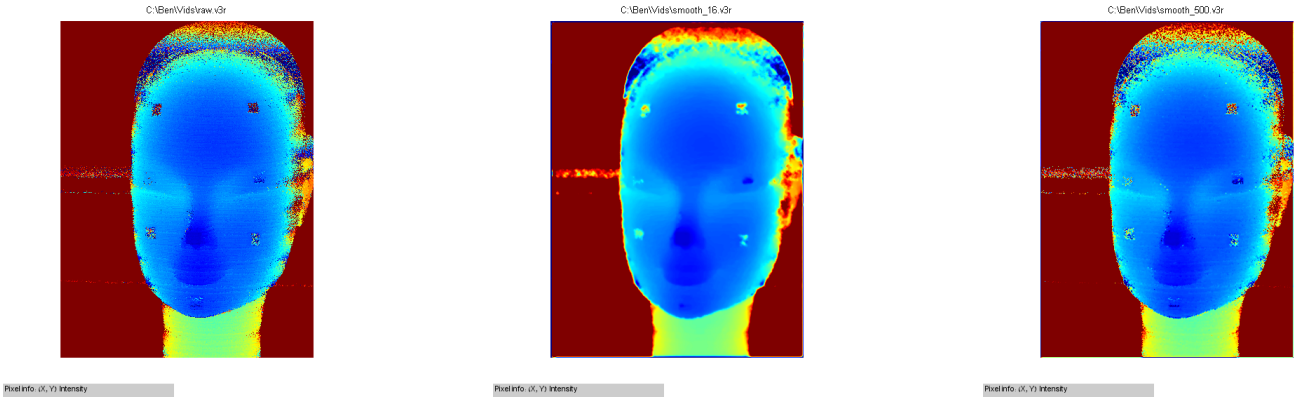
first row left to right: method (b); method (b) with 5x5 median filter

second row left to right: method (c); method (c) filtered with 5x5 median before TV regularization

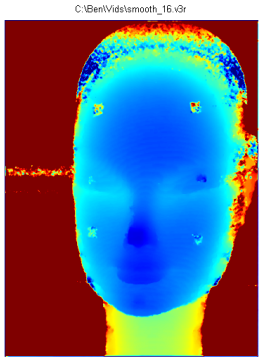
9.3 Comparing Different Constraints Parameters

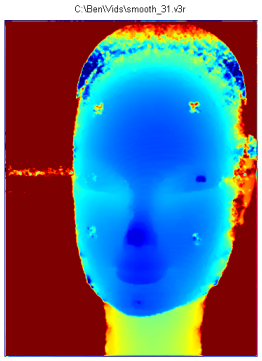


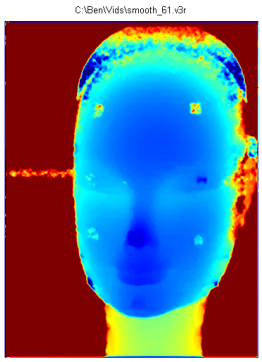
$L1 = 1$,$\alpha = 16$,$i = 12$, method (c)
from left to right: $r = 16$; $r = 31$; $r = 120$

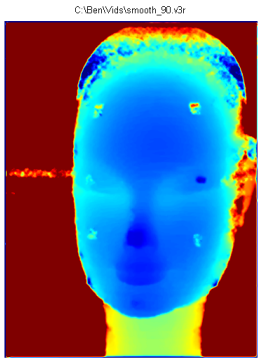


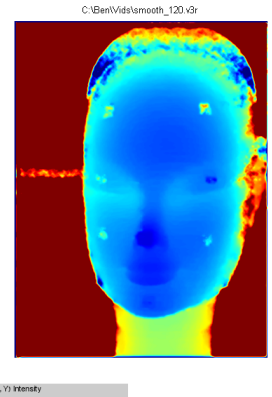
$L1 = 1$,$i = 12$,$r = 90$
from left to right: method (b); method (c) $\alpha = 16$; method (c) $\alpha = 500$









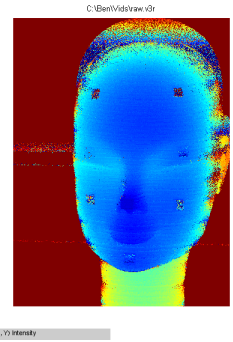


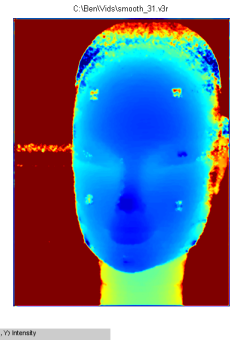
$L1 = 1$,$\alpha = 16$,$i = 12$, method (c)

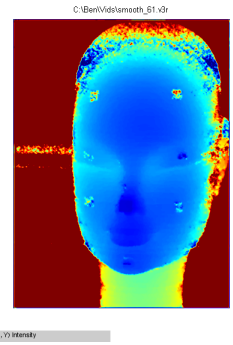
first row left to right: $r = 16$;$r = 31$

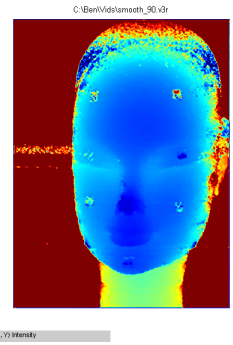
second row left to right: $r = 61$;$r = 90$

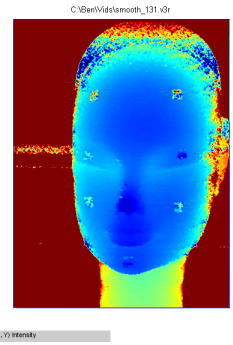
third row:$r = 120$

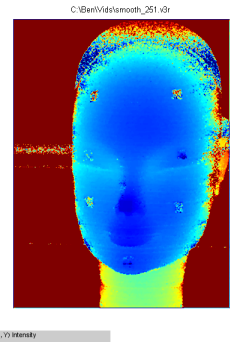


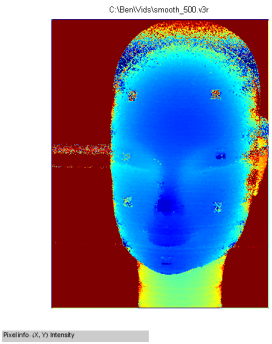












$L1 = 1, i = 12, r = 90$

first row left to right: method (b); method (c) $\alpha = 31$

second row left to right: method (c) $\alpha = 61; \alpha = 90$

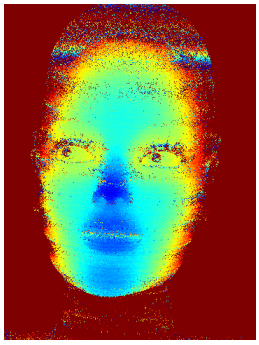
third row left to right: method (c) $\alpha = 131; \alpha = 251$

fourth row: method (c) $\alpha = 500$

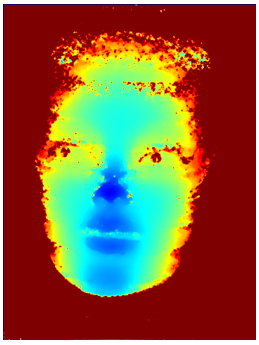
10 Possible Improvements

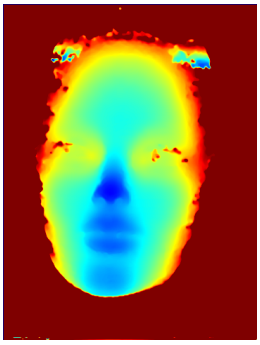
10.1 Artifacts Removal

Looking back at the results, we can see the effects of irregularities arising from the initial plane sweeping stage.



Depth map after initial plane sweep





Left: final reconstruction without per-filtering

Right: final reconstruction with median per-filtering

This proved to be even more problematic in cases with low SNR.

As seen in the above results, by using some filter after the initial sweep stage, we can significantly improve results.

Median filter was problematic in some cases so the choice of the appropriate filter remains.

(observe incorrect depth for sticker above right eye).

Since we do calculate relative probabilities, it should be possible to give some kind of a confidence value to each depth, ideally integrating that confidence value into some kind of a filter that will remove irregularities.

Therefore the new algorithm would change as follows:

1. initiate $\lambda^0 = 0$,$p^0 = 0$,$\mu^0 = 0$
2. per pixel x
   (a) pick $z(x)$ that minimizes 8
3. do i times
   (a) pass z through a filter considering confidence value. (New Step)
   (b) do $L1$ times
      i. optimize u using 15
      ii. optimize p using 12
      iii. update λ using 9
   (c) update μ using 14
   (d) update z using 18

10.2 Better Photo-consistency Model

After experimentation, we noticed that we can reduce the density of the planes sampled and compensate it with the alternating TV minimization.

Proving that our assumption for the surface prior holds even for bad initial depth estimation.

We hypothesize that inherently, there are other piece-wise properties of the objects being reconstructed.

Therefore if we are correct, it should be possible to assume a variety of related properties, for example piece-wise smoothness of albedo, piece-wise rigidness for moving objects etc.

References

[1] G. Rosman, A. Dubrovina, R. Kimmel, Sparse Modeling of Shape from Structured Light

[2] Chunlin Wu, Xue-Cheng Tai , AUGMENTED LAGRANGIAN METHOD, DUAL METHODS, AND SPLIT BREGMAN ITERATION FOR ROF, VECTORIAL TV, AND HIGH ORDER MODELS.

[3] Yilun Wang, Junfeng Yang, Wotao Yin, and Yin Zhang , A NEW ALTERNATING MINIMIZATION ALGORITHM FOR TOTAL VARIATION IMAGE RECONSTRUCTION.

[4] http://www.brnt.eu/phd/node16.html

[5] J. P. Tardif and S. Roy. A MRF formulation for coded structured light.

[6] O. Rubinstein, Y. Honen, A. M. Bronstein, M. M. Bronstein, and R. Kimmel, 3D color video camera, In Proc. of Workshop on 3D Digital Imaging and Modeling (3DIM), 2009.

[7] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT10/

[8] multiple view geometry in computer vision by R. Hartley , A. Zisserman

[9] http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[10] V. Volkov, Better performance at lower occupancy, at Proceedings of the GPU Technology Conference 2010