习题课 2022.09.14

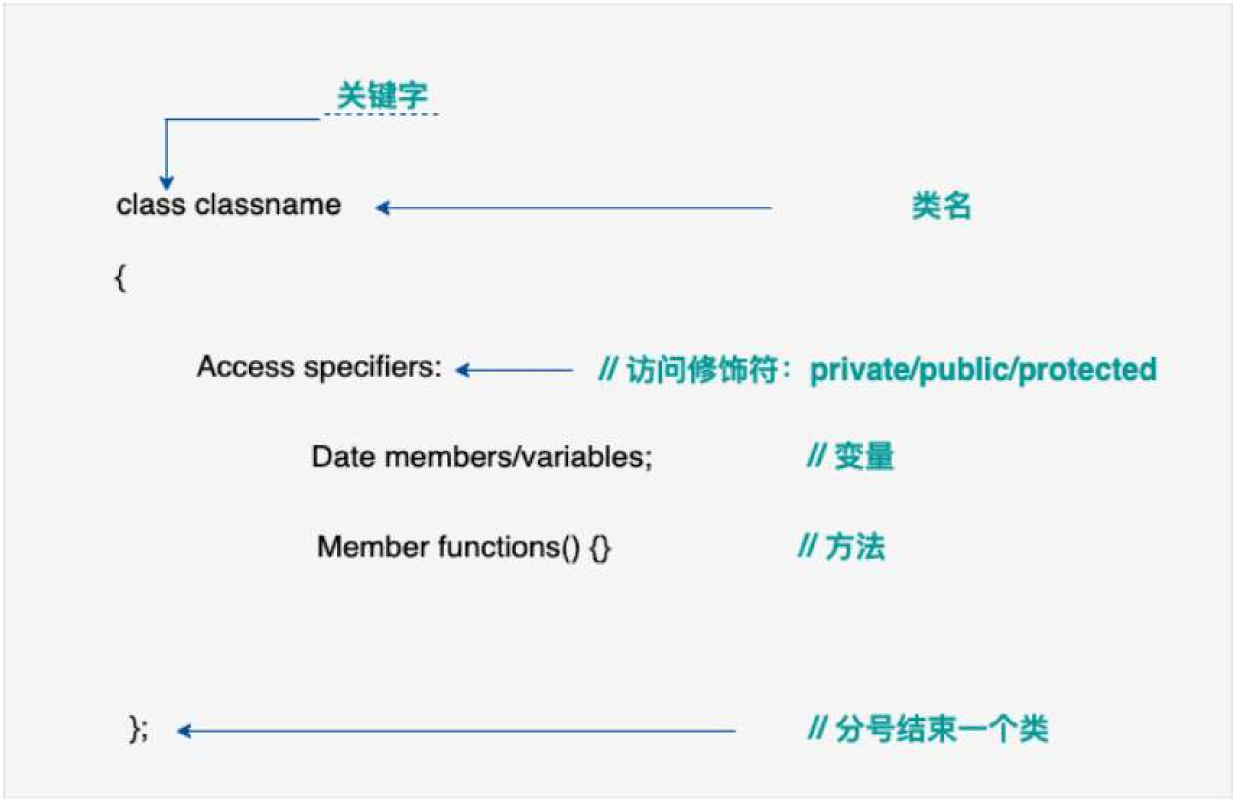
1. 认识到自己日后想做什么？数值解 or CS（CG or CV or ...）。
    - 配环境是常态，自己鼓捣，**错的多了**、见识的多了就好了
2. Markdown 不是花时间学出来的，直接上手敲。
    - Latex 是论文需要的，推荐线上 Overleaf ，即写即得、无需配环境；
    - Markdown 更介于 Latex 和 Word 折合，更强大和简便；
3. C++ 简单介绍 *类 & 封装 & 继承 & 多态*。
    - 用多少、学多少、不要捧着一本书看完才动手编程或看代码！！！
4. 程序简单说明。

C++ 类 & 对象

- C++ 在 C 语言的基础上增加了面向对象编程，C++ 支持面向对象程序设计。类是 C++ 的核心特性，通常被称为用户定义的类型。
- 类用于指定对象的形式，它包含了数据表示法和用于处理数据的方法。类中的数据和方法称为类的成员。函数在一个类中被称为类的成员。

C++ 类定义

- 定义一个类，本质上是定义一个数据类型的蓝图，定义了类的对象包括了什么，以及可以在这个对象上执行哪些操作。

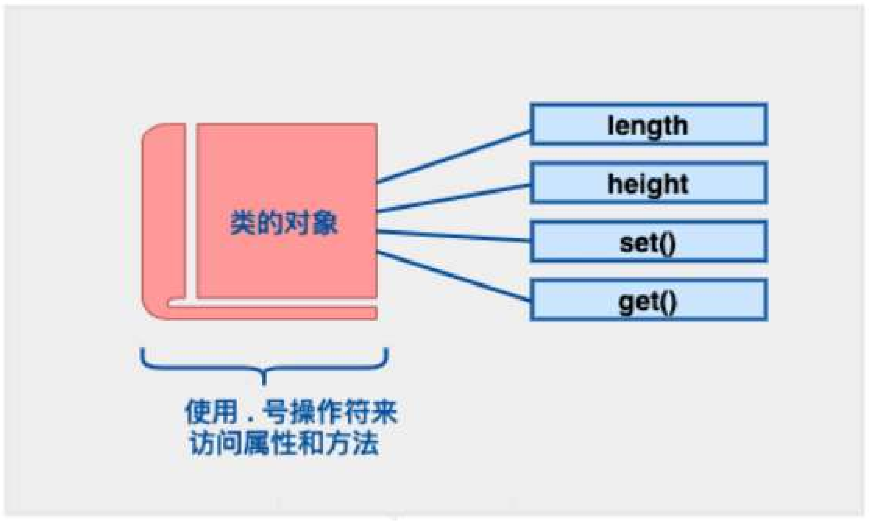


- 类定义后必须跟着一个分号或一个声明列表。例如，我们使用关键字 **class** 定义 Box 数据类型，如下所示：

```
class Box
{
        public:
                double length; // 盒子的长度
                double breadth; // 盒子的宽度
                double height; // 盒子的高度

            // 成员函数声明
            double get(void);
            void set( double len, double bre, double hei );
    };

    // 成员函数定义
    double Box::get(void)
    {
            return length * breadth * height;
    }

    void Box::set( double len, double bre, double hei)
    {
            length = len; breadth = bre; height = hei;
    }

    int main( )
    {
            Box Box1; // 声明 Box1，类型为 Box
            Box Box3; // 声明 Box3，类型为 Box
            double volume = 0.0; // 用于存储体积

            // box 1 详述
            Box1.height = 5.0;
            Box1.length = 6.0;
            Box1.breadth = 7.0;

            // box 1 的体积
            volume = Box1.height * Box1.length * Box1.breadth;
            cout << "Box1 的体积：" << volume <<endl;

            // box 3 详述
            Box3.set(16.0, 8.0, 12.0);
            volume = Box3.get();
            cout << "Box3 的体积：" << volume <<endl;

            return 0;
    }
```

- 关键字 public 确定了类成员的访问属性。在类对象作用域内，公共成员在类的外部是可访问的。

访问数据成员



- 需要注意的是，私有的成员和受保护的成员不能使用直接成员访问运算符 （.） 来直接访问。

类 & 对象详解

概念	描述
类成员函数	类的成员函数是指那些把定义和原型写在类定义内部的函数，就像类定义中的其他变量一样。
类访问修饰符	类成员可以被定义为 public、private 或 protected。默认情况下是定义为 private。
构造函数 & 析构函数	类的构造函数是一种特殊的函数，在创建一个新的对象时调用。类的析构函数也是一种特殊的函数，在删除所创建的对象时调用。
拷贝构造函数	拷贝构造函数，是一种特殊的构造函数，它在创建对象时，是使用同一类中之前创建的对象来初始化新创建的对象。
友元函数	**友元函数**可以访问类的 private 和 protected 成员。
内联函数	通过内联函数，编译器试图在调用函数的地方扩展函数体中的代码。
this 指针	每个对象都有一个特殊的指针 this，它指向对象本身。
指向类的指针	指向类的指针方式如同指向结构的指针。实际上，类可以看成是一个带有函数的结构。
类的静态成员	类的数据成员和函数成员都可以被声明为静态的。

C++ 数据封装

所有的 C++ 程序都有以下两个基本要素：

- **程序语句（代码）：** 这是程序中执行动作的部分，它们被称为函数。
- **程序数据：** 数据是程序的信息，会受到程序函数的影响。

封装是面向对象编程中的把数据和操作数据的函数绑定在一起的一个概念，这样能避免受到外界的干扰和误用，从而确保了安全。数据封装引申出了另一个重要的 OOP 概念，即**数据隐藏**。

数据封装是一种把数据和操作数据的函数捆绑在一起的机制，**数据抽象**是一种仅向用户暴露接口而把具体的实现细节隐藏起来的机制。

C++ 通过创建**类**来支持封装和数据隐藏（public、protected、private）。我们已经知道，类包含私有成员（private）、保护成员（protected）和公有成员（public）成员。默认情况下，在类中定义的所有项目都是私有的。例如：

```
class Box
{
        public:
                double getVolume(void) { return length * breadth * height; }
        private:
                double length; // 长度
                double breadth; // 宽度
                double height; // 高度
};
```

变量 length、breadth 和 height 都是私有的（private）。这意味着它们只能被 Box 类中的其他成员访问，而不能被程序中其他部分访问。这是实现封装的一种方式。

为了使类中的成员变成公有的（即，程序中的其他部分也能访问），必须在这些成员前使用 **public** 关键字进行声明。所有定义在 public 标识符后边的变量或函数可以被程序中所有其他的函数访问。

把一个类定义为另一个类的友元类，会暴露实现细节，从而降低了封装性。理想的做法是尽可能地对外隐藏每个类的实现细节。

```
#include <iostream>
using namespace std;
class Adder
{
        public:
        // 构造函数
        Adder(int i = 0) { total = i; }
        // 对外的接口
        void addNum(int number) { total += number; }
        // 对外的接口
        int getTotal() { return total; }; private:
        // 对外隐藏的数据
        int total;
};

int main( )
{
        Adder a;
        a.addNum(10);
        a.addNum(20);
        a.addNum(30);
        cout << "Total " << a.getTotal() <<endl; return 0;
}
```

- 上面的类把数字相加，并返回总和。公有成员 **addNum** 和 **getTotal** 是对外的接口，用户需要知道它们以便使用类。私有成员 **total** 是对外隐藏的，用户不需要了解它，但它又是类能正常工作所必需的。

C++ 继承

- 继承允许我们依据另一个类来定义一个类，这使得创建和维护一个应用程序变得更容易。这样做，也达到了重用代码功能和提高执行效率的效果。
- 当创建一个类时，您不需要重新编写新的数据成员和成员函数，只需指定新建的类继承了一个已有的类的成员即可。这个已有的类称为**基类**，新建的类称为**派生类**。 `class derived-class: access-specifier base-class`

其中，访问修饰符 access-specifier 是 **public、protected** 或 **private** 其中的一个，base-class 是之前定义过的某个类的名称。如果未使用访问修饰符 access-specifier，则默认为 private。

- 继承代表了 **is a** 关系。例如，哺乳动物是动物，狗是哺乳动物，因此，狗是动物，等等。

```
#include <iostream>
using namespace std;

// 基类
class Shape
{
        public:
        void setWidth(int w) { width = w; }
        void setHeight(int h) { height = h; }
        protected:
                int width;
                int height;
};

// 派生类
class Rectangle: public Shape
{
        public:
                int getArea() { return (width * height); }
};

int main(void)
{
        Rectangle Rect;
        Rect.setWidth(5);
        Rect.setHeight(7);
        // 输出对象的面积
        cout << "Total area: " << Rect.getArea() << endl; return 0;
}
```

C++ 多态

- **多态**按字面的意思就是多种形态。当类之间存在层次结构，并且类之间是通过继承关联时，就会用到多态。
- C++ 多态意味着调用成员函数时，会根据调用函数的对象的类型来执行不同的函数。
- 下面的实例中，基类 Shape 被派生为两个类，如下所示：

```
#include <iostream>
using namespace std;
class Shape
{
        protected:
        int width, height;
        public:
        Shape( int a=0, int b=0) { width = a; height = b; }
        int area() { cout << "Parent class area :" <<endl; return 0; }
};
class Rectangle: public Shape
{
        public:
                Rectangle( int a=0, int b=0):Shape(a, b) { }
                int area () { cout << "Rectangle class area :" <<endl; return (width * height); }
};
class Triangle: public Shape
{
        public: Triangle( int a=0, int b=0):Shape(a, b) { }
        int area () { cout << "Triangle class area :" <<endl; return (width * height / 2); }
};

// 程序的主函数
int main( )
{
        Shape *shape;
        Rectangle rec(10,7);
        Triangle tri(10,5);

        // 存储矩形的地址
        shape = &rec;
        // 调用矩形的求面积函数
        area shape->area();
        // 存储三角形的地址
        shape = &tri;
        // 调用三角形的求面积函数
        area shape->area();

        return 0;
}
```

输出:

```
Parent class area :
Parent class area :
```

导致错误输出的原因是，调用函数 area() 被编译器设置为基类中的版本，这就是所谓的**静态多态**，或**静态链接** - 函数调用在程序执行前就准备好了。有时候这也被称为**早绑定**，因为 area() 函数在程序编译期间就已经设置好了。

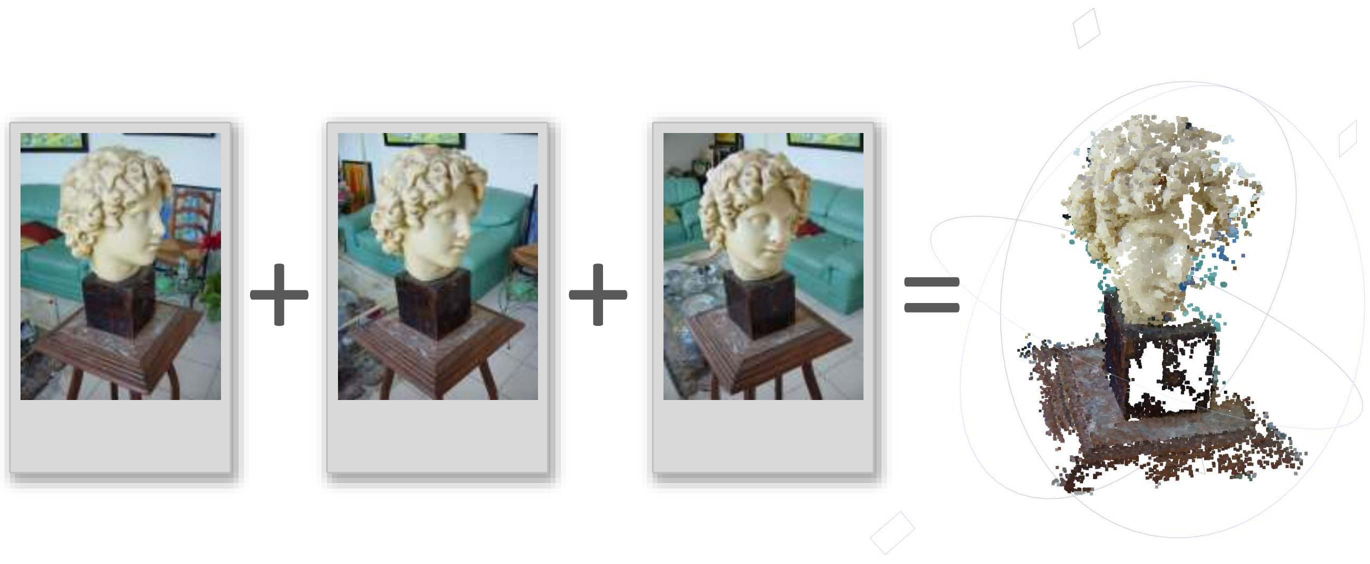
但现在，让我们对程序稍作修改，在 Shape 类中，area() 的声明前放置关键字 **virtual**，如下所示:

```
class Shape
{
        protected:
                int width, height;
        public:
                Shape( int a=0, int b=0) { width = a; height = b; }
                virtual int area() { cout << "Parent class area :" <<endl; return 0; }
};
```

输出：

```
Rectangle class area :
Triangle class area :
```

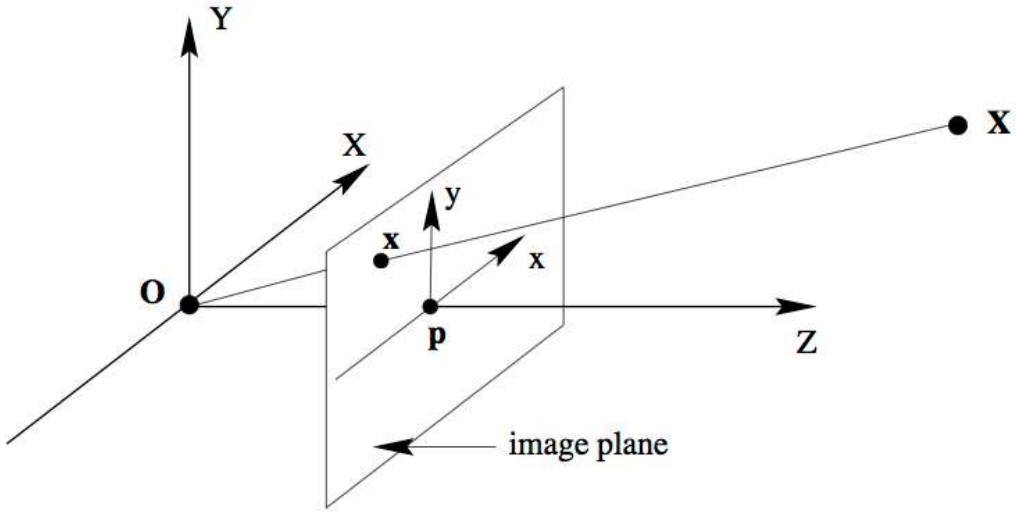
- 此时，编译器看的是指针的内容，而不是它的类型。因此，由于 tri 和 rec 类的对象的地址存储在 *shape 中，所以会调用各自的 area() 函数。*
- 每个子类都有一个函数 area() 的独立实现。这就是**多态**的一般使用方式。
- 有了多态，就可有多个不同的类，都带有同一个名称但具有不同实现的函数，函数的参数甚至可以是相同的。
- **虚函数** 是在基类中使用关键字 **virtual** 声明的函数。在派生类中重新定义基类中定义的虚函数时，会告诉编译器不要静态链接到该函数。我们想要的是在程序中任意点可以根据所调用的对象类型来选择调用的函数，这种操作被称为**动态链接**，或**后期绑定**。



Multi-view 3D Reconstruction

2022.10.13
杨乐园

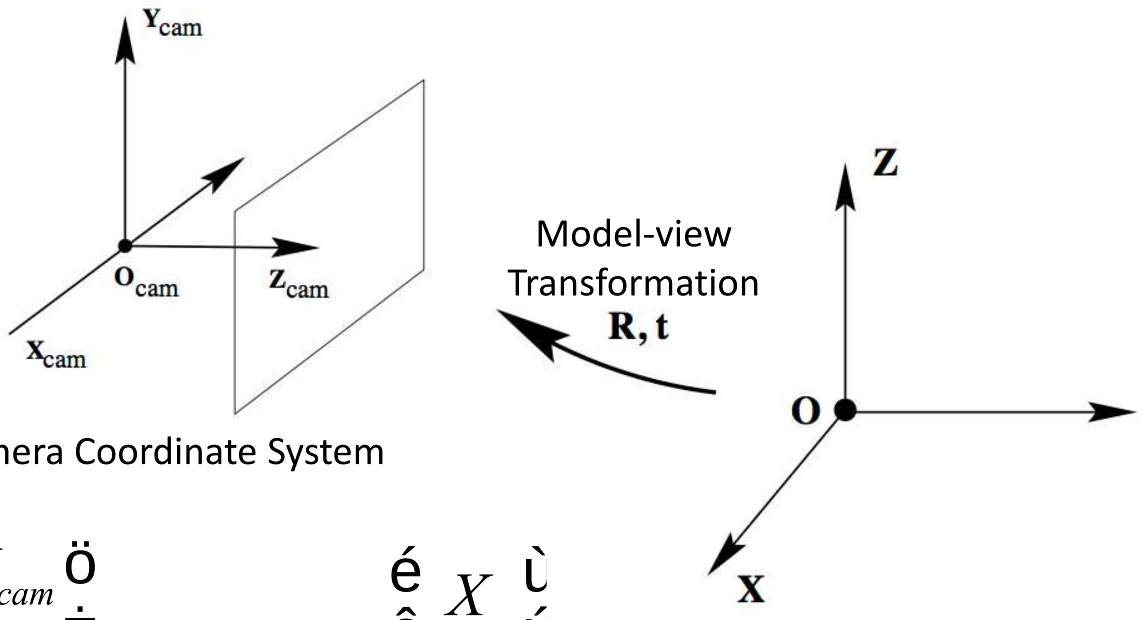
When they take a picture:



$$/ \begin{bmatrix} x \\ y \\ f \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \Longleftrightarrow \quad \begin{bmatrix} x \\ y \\ f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Up to a scale

In a real world



Camera Coordinate System

World Coordinate System

$$\begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

World Coor. → Camera Coor.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\Updownarrow$$

$$\mathbf{x} = \mathbf{K}\left[\mathbf{R} \middle| \mathbf{t}\right]\mathbf{X}$$

$$\Updownarrow$$

$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

Camera Parameter
Camera Projection Matrix $\mathbf{P} = \underbrace{\mathbf{K}}_{\text{Intrinsic}}\underbrace{\left[\mathbf{R} \middle| \mathbf{t}\right]}_{\text{Extrinsic}}$

Other Intrinsic Camera Parameters

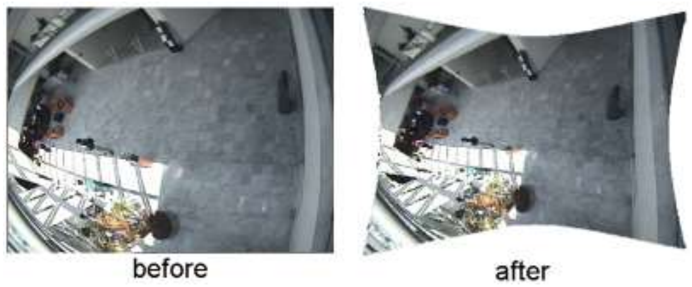
- Principle point offset
  - especially when images are cropped (Internet)
- Skew

$$\mathbf{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \mathbf{K} = \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

- Radial distortion (due to optics of the lens)

$$r^2 = \|\mathbf{x}\|^2 = x^2 + y^2$$

$$\mathbf{x}' = (1 + k_1 r^2 + k_2 r^4)\mathbf{x}$$



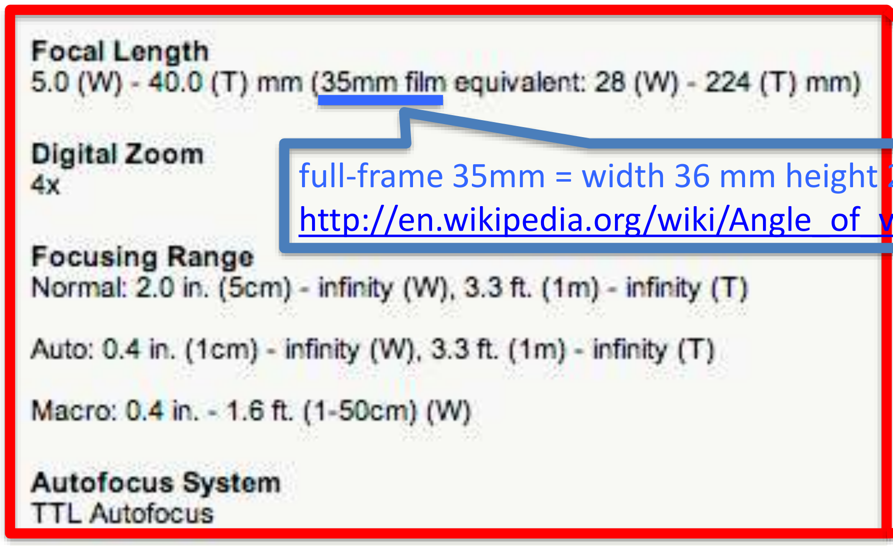
White-board Exercise

$$\mathbf{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

I brought a normal compact digital camera.

I took a picture at resolution 640x480 pixels.

What is the possible value of the focal length f?



Focal Length
5.0 (W) - 40.0 (T) mm (35mm film equivalent: 28 (W) - 224 (T) mm)

Digital Zoom
4x

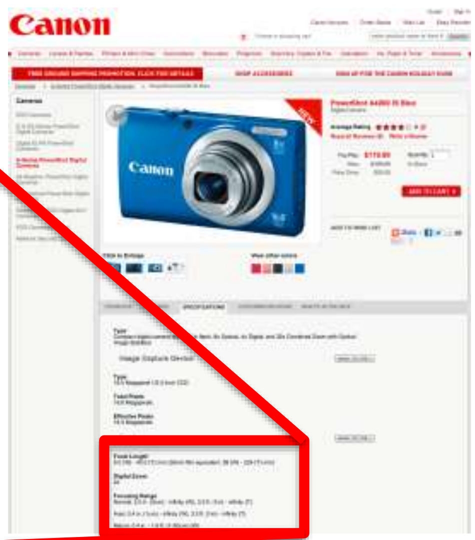
Focusing Range
Normal: 2.0 in. (5cm) - infinity (W), 3.3 ft. (1m) - infinity (T)

Auto: 0.4 in. (1cm) - infinity (W), 3.3 ft. (1m) - infinity (T)

Macro: 0.4 in. - 1.6 ft. (1-50cm) (W)

Autofocus System
TTL Autofocus

full-frame 35mm = width 36 mm height 24 mm
http://en.wikipedia.org/wiki/Angle_of_view



http://goo.gl/ENaw4

Wait: How to get the focal length?

- Auto-calibration

  Self-Calibration and Metric Reconstruction in spite of Varying and Unknown Internal Camera Parameters, M Pollefeys, R Koch and L Van Gool, 1998.
  http://mit.edu/jxiao/Public/software/autocalibrate/autocalibration_lin.m

- Grid Search to look for the solution with minimal reprojection error

  ```
  for f=min_f:max_f
      do everything, then obtain reprojection error after bundle adjustment
  ```

- Optimize for this value in bundle adjustment
- Camera Calibration (with checkerboard)

  http://www.vision.caltech.edu/bouguetj/calib_doc/

- EXIF of JPEG file recorded from digital camera

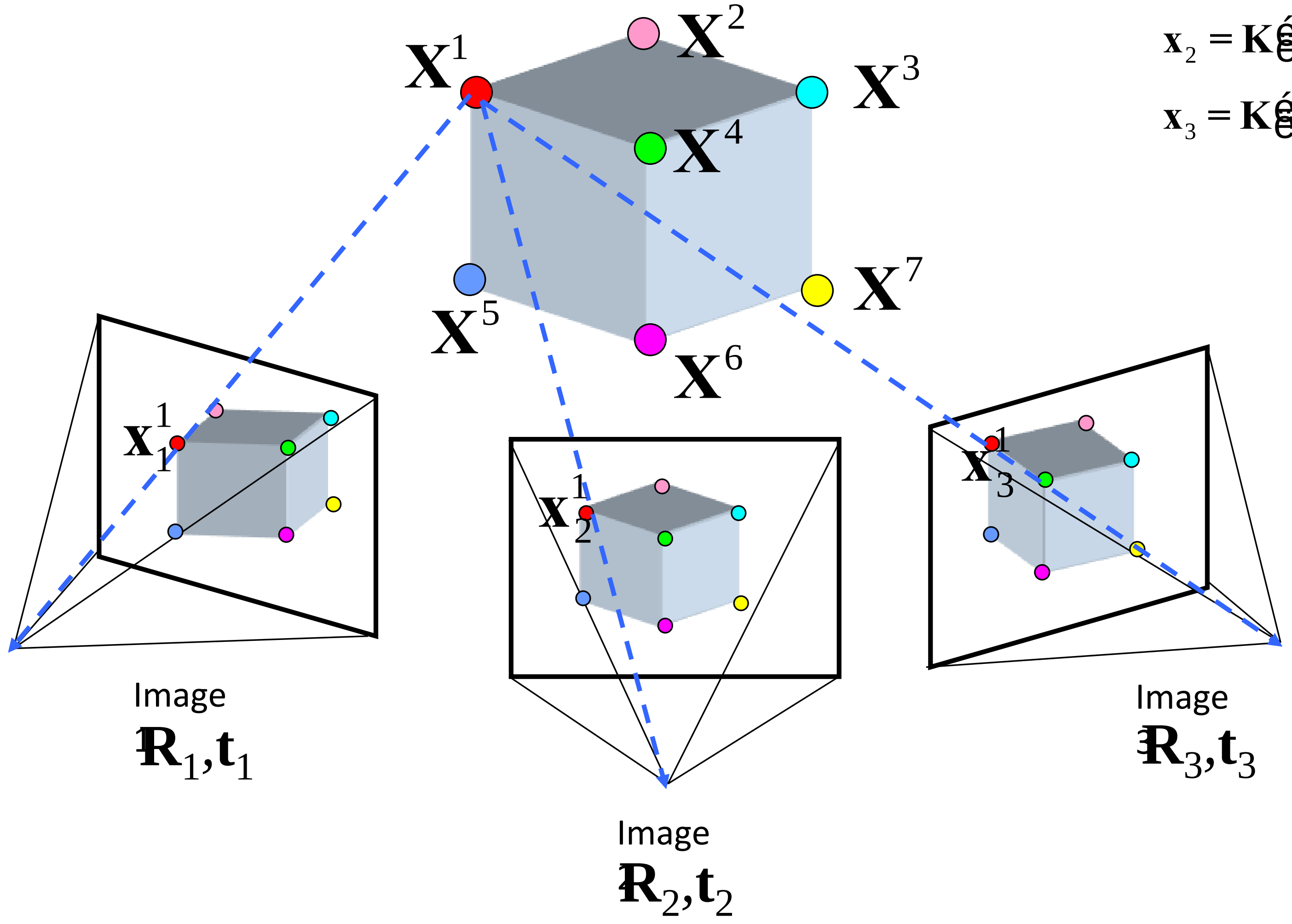
  Read the code of Bundler to understand how to convert EXIF into focal length value
  http://phototour.cs.washington.edu/bundler/

- When people take three pictures with same camera setting:

$$\mathbf{x}_1 = \mathbf{K}\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right]\mathbf{X}$$

$$\mathbf{x}_2 = \mathbf{K}\left[\mathbf{R}_2 \middle| \mathbf{t}_2\right]\mathbf{X}$$

$$\mathbf{x}_3 = \mathbf{K}\left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]\mathbf{X}$$

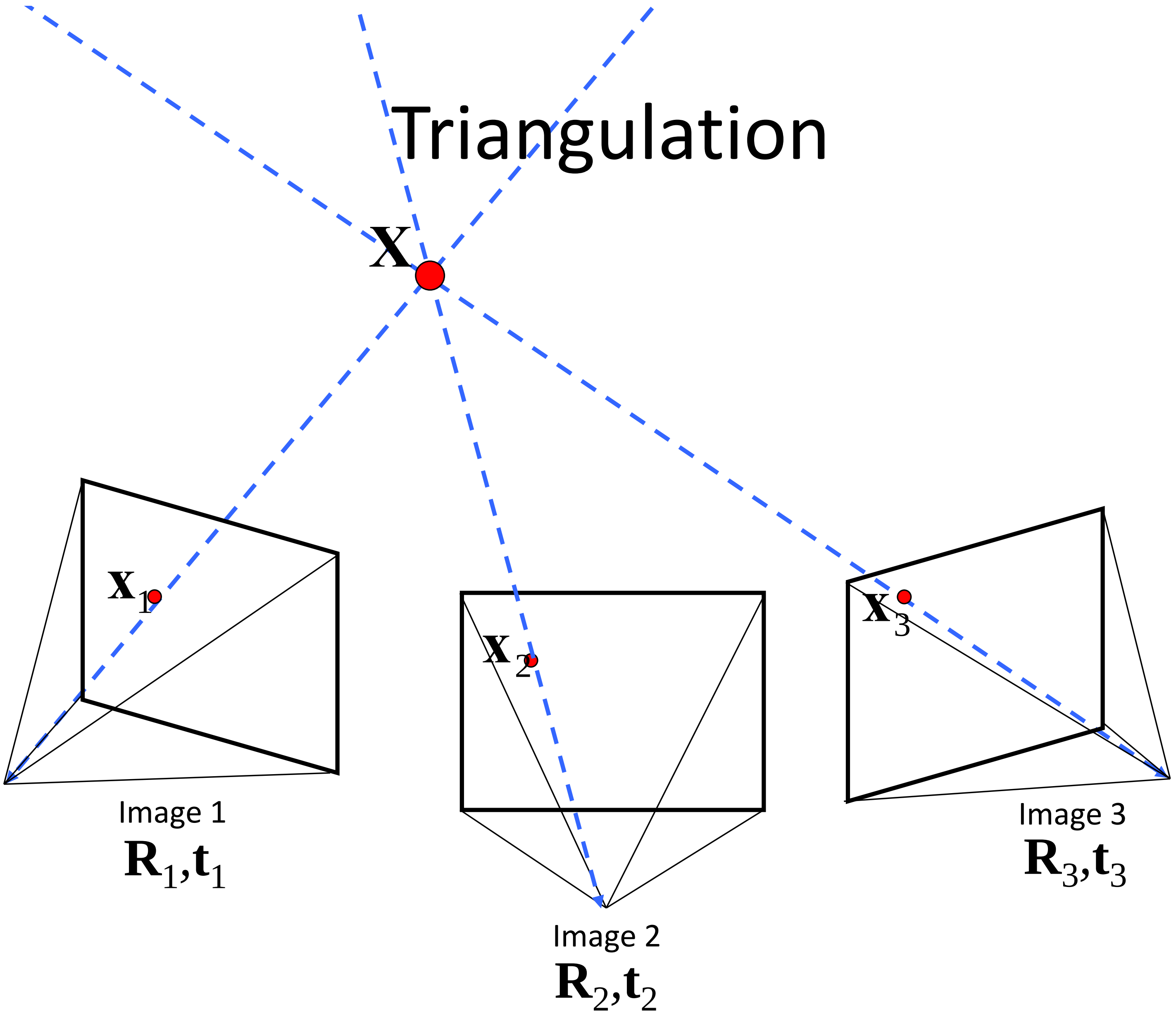


- When people take three pictures with same camera setting:

	Point 1	Point 2	Point 3
Image 1	$\mathbf{x}_1^1 = \mathbf{K}\left[\mathbf{R}_1 \mid \mathbf{t}_1\right]\mathbf{X}^1$	$\mathbf{x}_1^2 = \mathbf{K}\left[\mathbf{R}_1 \mid \mathbf{t}_1\right]\mathbf{X}^2$	
Image 2	$\mathbf{x}_2^1 = \mathbf{K}\left[\mathbf{R}_2 \mid \mathbf{t}_2\right]\mathbf{X}^1$	$\mathbf{x}_2^2 = \mathbf{K}\left[\mathbf{R}_2 \mid \mathbf{t}_2\right]\mathbf{X}^2$	$\mathbf{x}_2^3 = \mathbf{K}\left[\mathbf{R}_2 \mid \mathbf{t}_2\right]\mathbf{X}^3$
Image 3	$\mathbf{x}_3^1 = \mathbf{K}\left[\mathbf{R}_3 \mid \mathbf{t}_3\right]\mathbf{X}^1$		$\mathbf{x}_3^3 = \mathbf{K}\left[\mathbf{R}_3 \mid \mathbf{t}_3\right]\mathbf{X}^3$

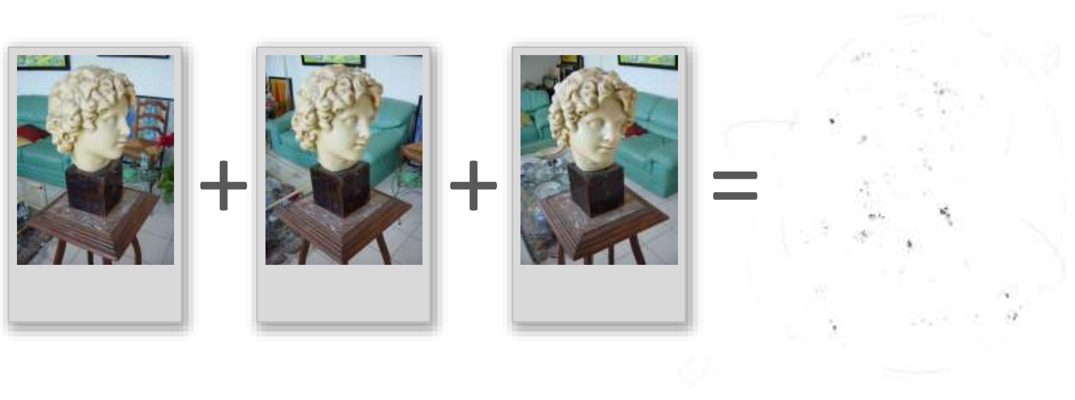
Same Camera Same Setting = Same $\mathbf{K}$

Triangulation


$\mathbf{X}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
Image 1
$\mathbf{R}_1,\mathbf{t}_1$
Image 2
$\mathbf{R}_2,\mathbf{t}_2$
Image 3
$\mathbf{R}_3,\mathbf{t}_3$

Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling

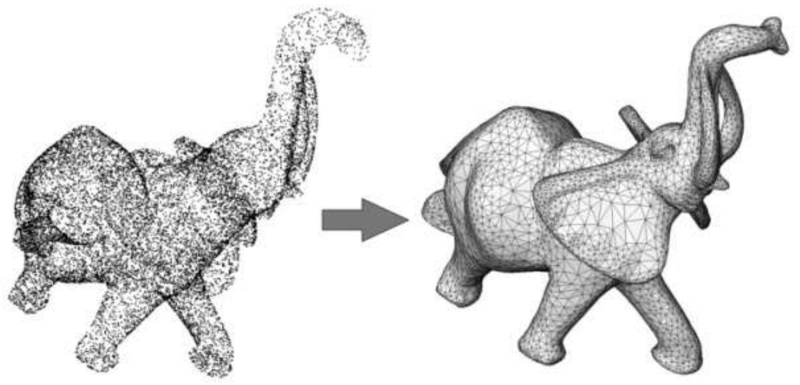


Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling

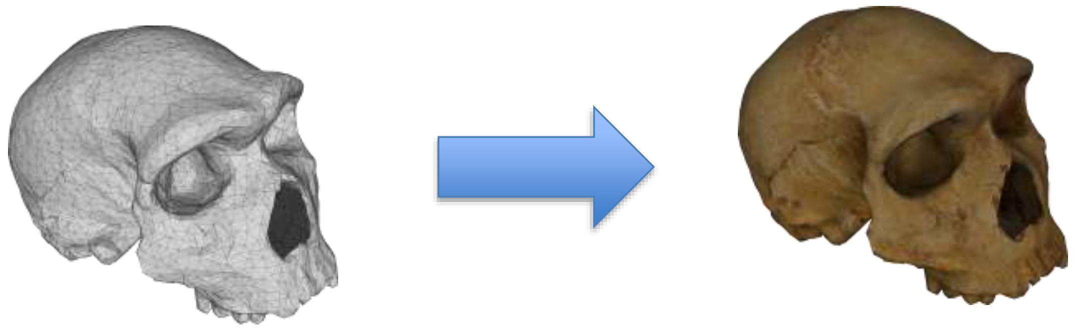
Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



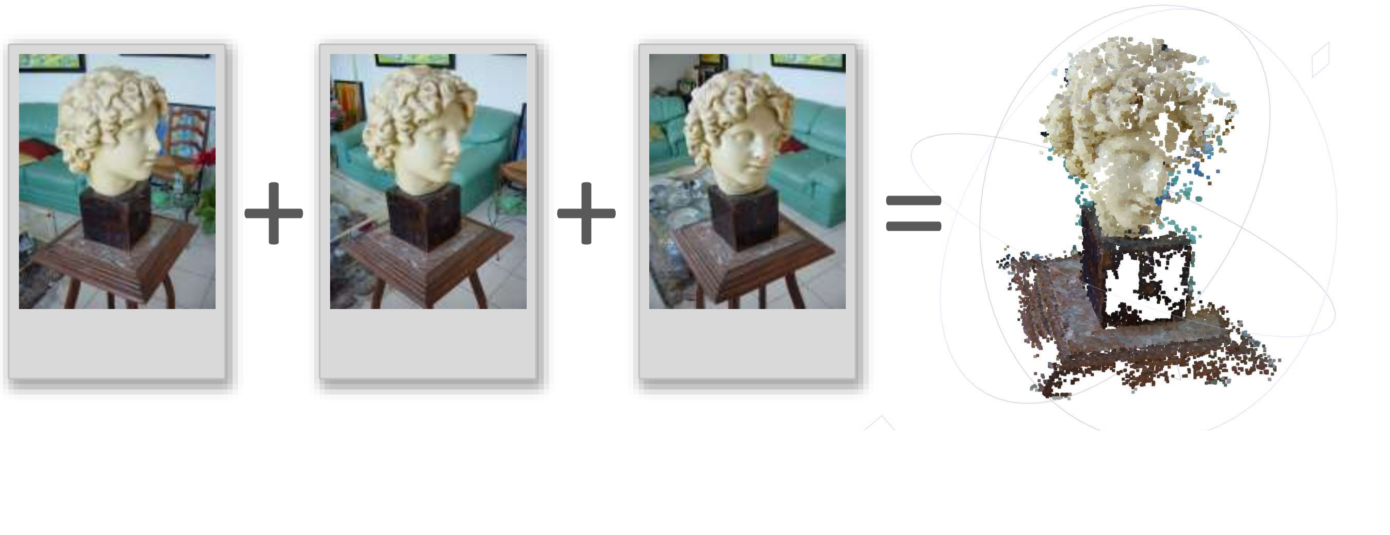
Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



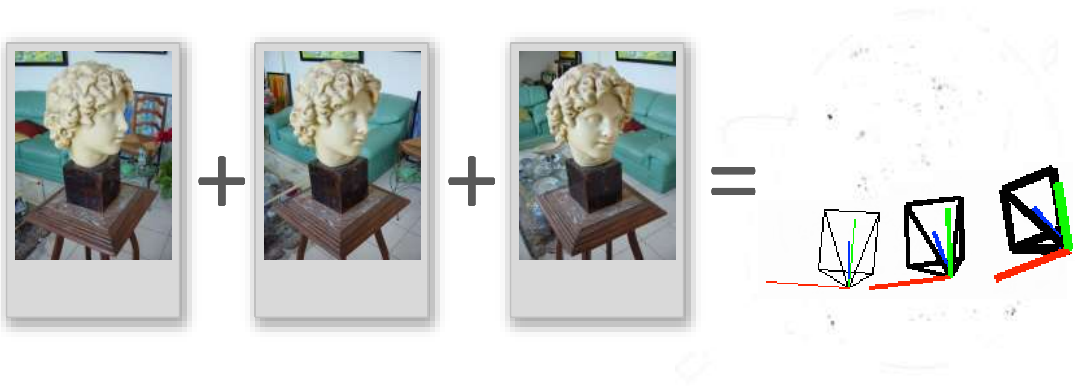
Finished!

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



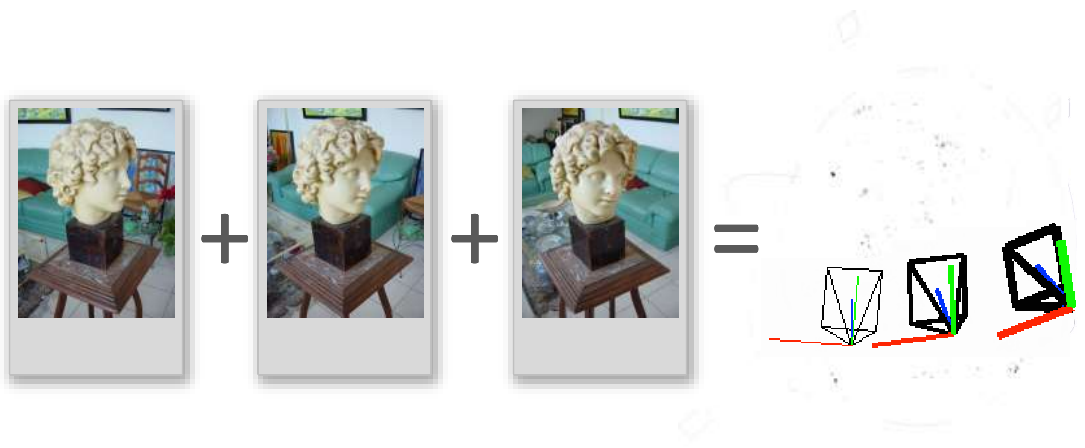
Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling

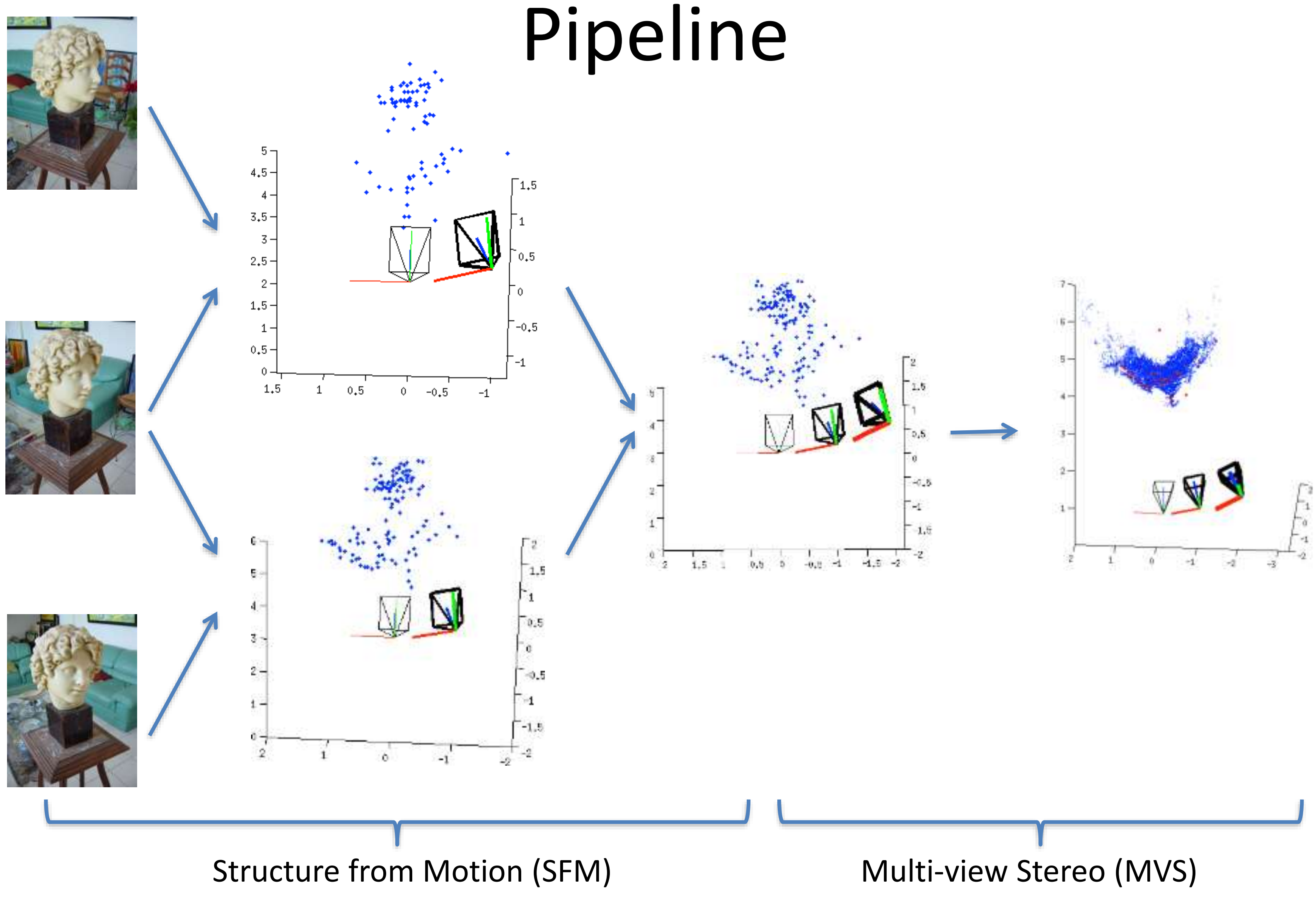


Structure From Motion

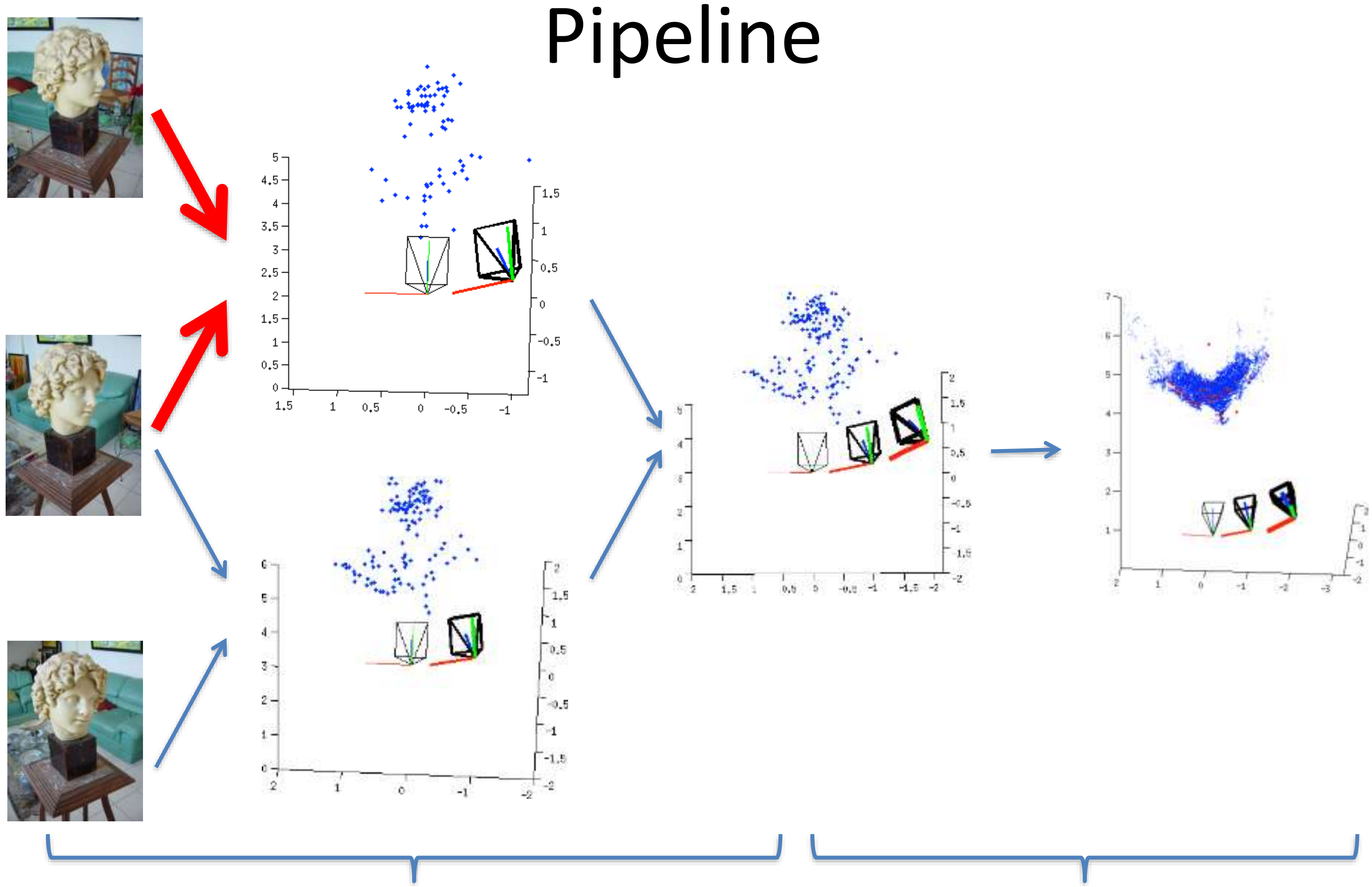
- Structure = 3D Point Cloud of the Scene
- Motion = Camera Location and Orientation
- SFM = Get the Point Cloud from Moving Cameras
- Structure and Motion: Joint Problems to Solve



Pipeline



Pipeline

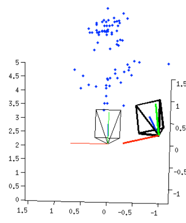


Structure from Motion (SFM)

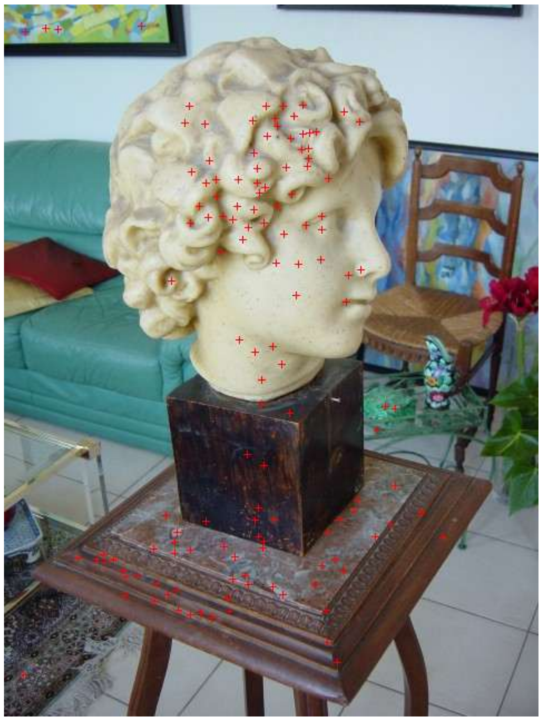
Multi-view Stereo (MVS)

Two-view Reconstruction

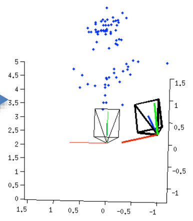
$$\begin{tikzcd}
\text{keypoints} \arrow[dr] & & & & & \\
& \text{match} \arrow[r] & \text{fundamental matrix} \arrow[r] & \text{essential matrix} \arrow[r] & {[R|t]} \arrow[r] & \text{triangulation} \arrow[r] & {} \\
\text{keypoints} \arrow[ur] & & & & &
\end{tikzcd}$$



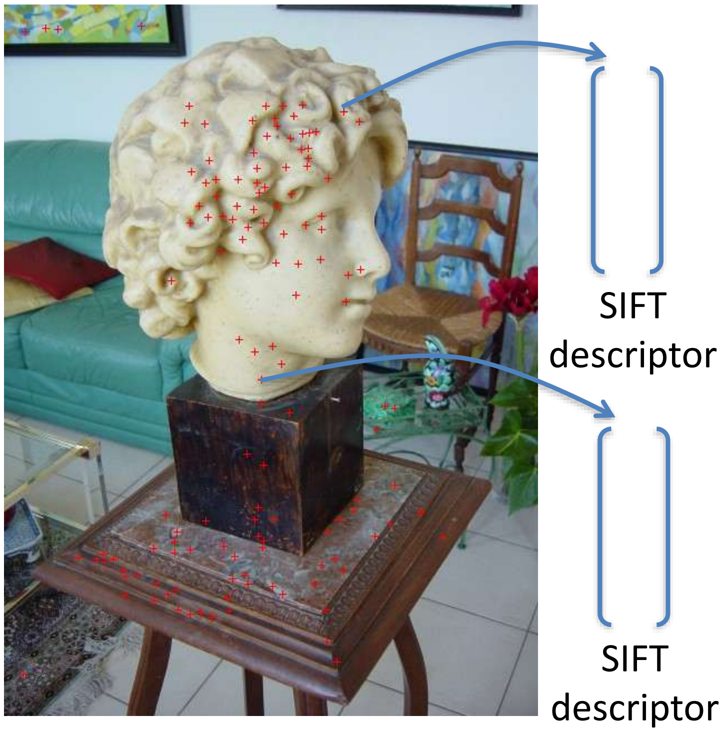
Keypoints Detection

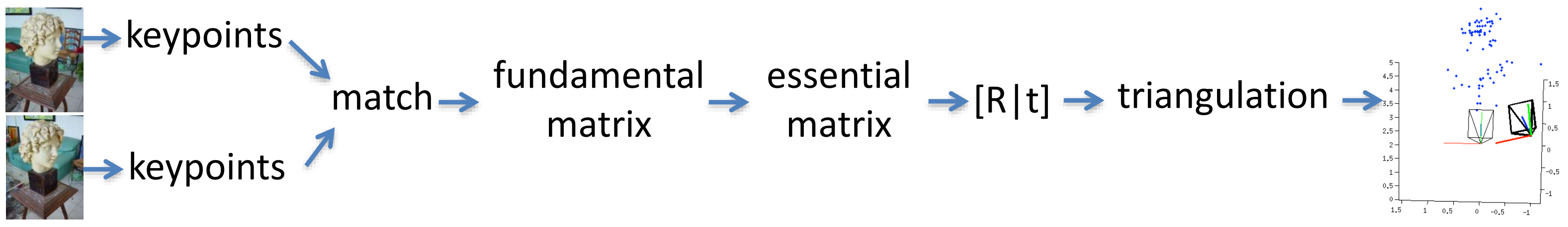


$$\begin{tikzcd}
\text{image} \arrow[r] & \text{keypoints} \arrow[dr] & & & & & & \\
& & \text{match} \arrow[r] & \text{fundamental matrix} \arrow[r] & \text{essential matrix} \arrow[r] & {[R|t]} \arrow[r] & \text{triangulation} \arrow[r] & \text{3D} \\
\text{image} \arrow[r] & \text{keypoints} \arrow[ur] & & & & & &
\end{tikzcd}$$

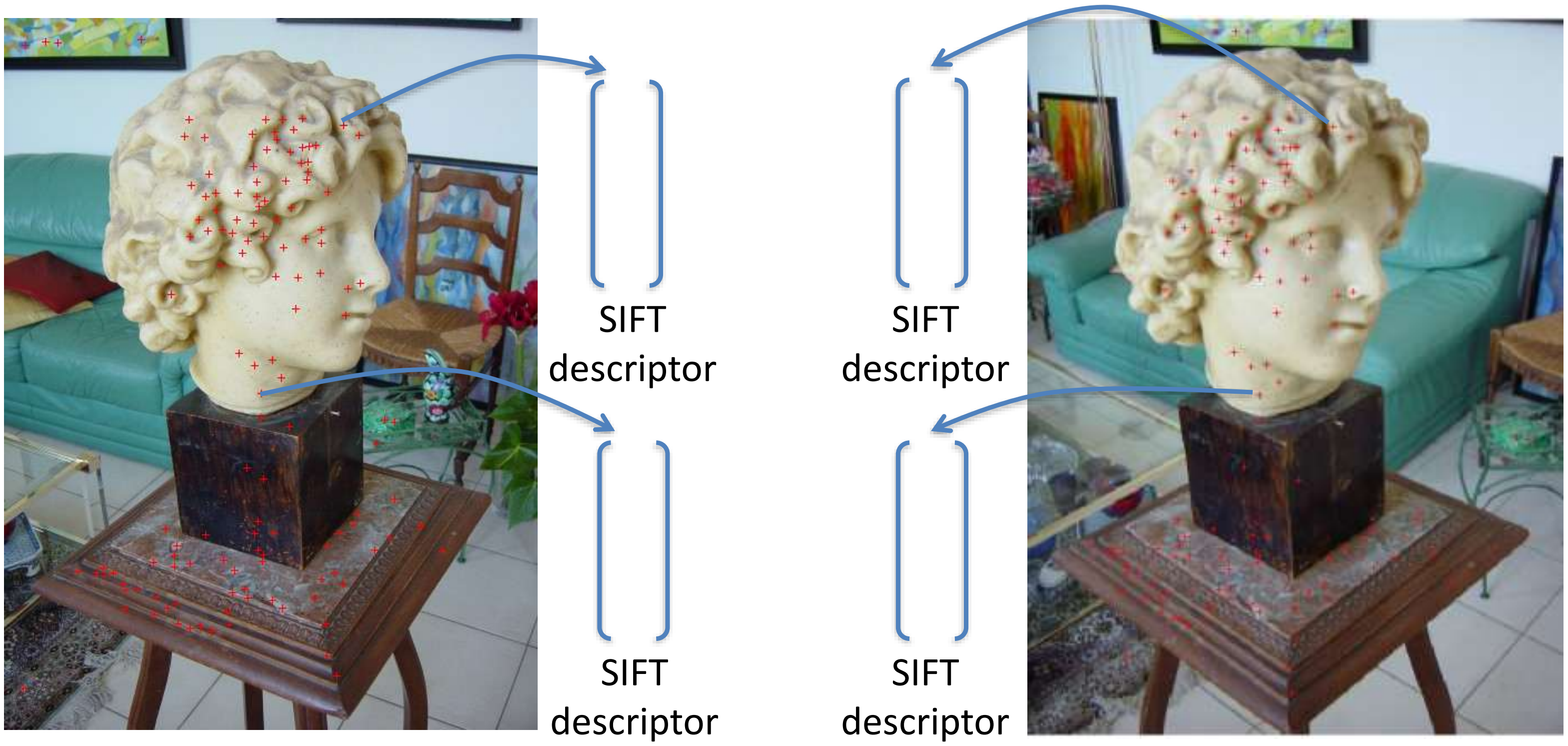


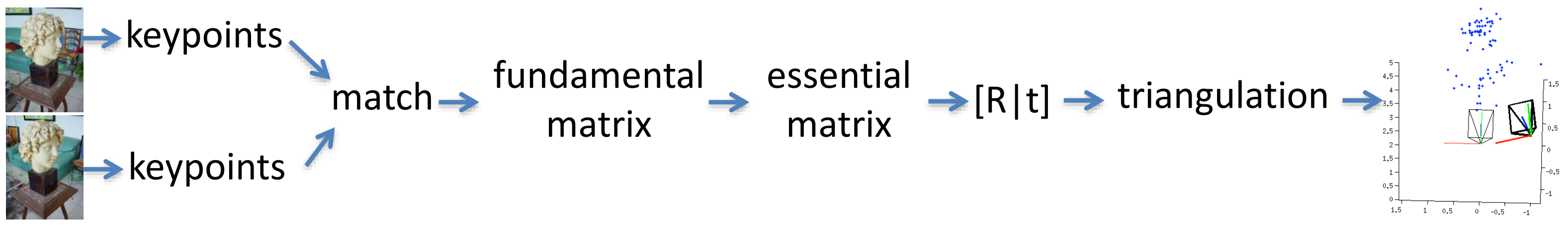
Descriptor for each point



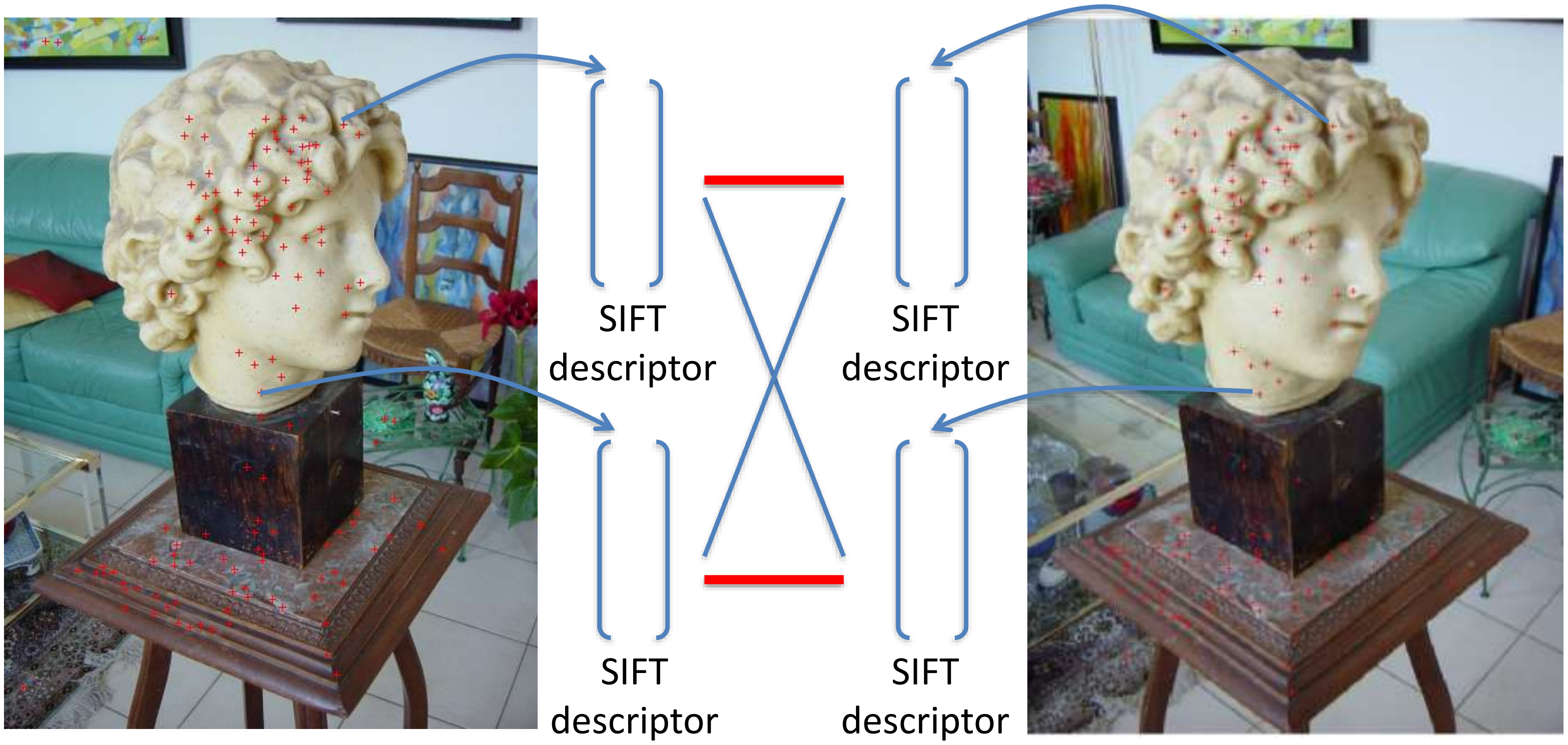


Same for the other images

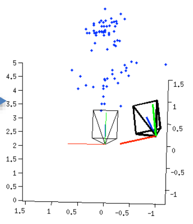




Point Match for correspondences



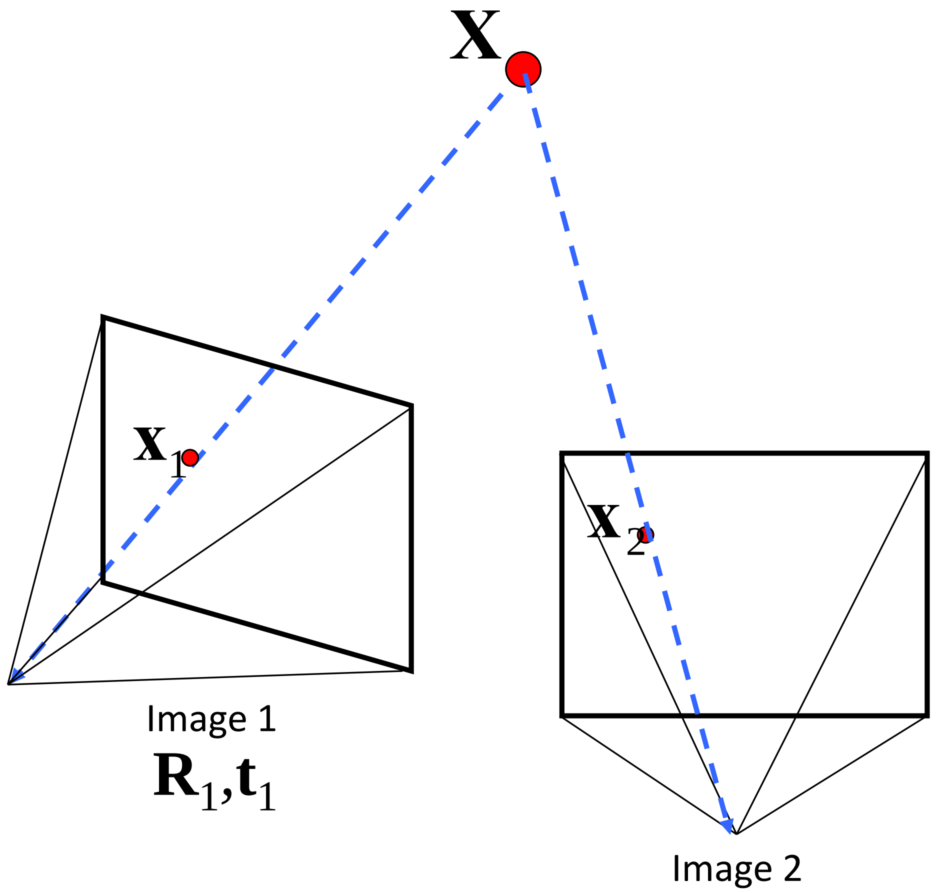
$$\begin{tikzcd}
\text{keypoints} \arrow[dr] & & & & & \\
 & \text{match} \arrow[r] & \text{fundamental matrix} \arrow[r] & \text{essential matrix} \arrow[r] & {[R|t]} \arrow[r] & \text{triangulation} \\
\text{keypoints} \arrow[ur] & & & & &
\end{tikzcd}$$



$$\mathbf{x}_1 = \mathbf{K}\left[\mathbf{R}_1 | \mathbf{t}_1\right]\mathbf{X}$$

$$\mathbf{x}_2 = \mathbf{K}\left[\mathbf{R}_2 | \mathbf{t}_2\right]\mathbf{X}$$

Fundamental Matrix



$$\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$$

$$\mathbf{x}_1^T \mathbf{F} \mathbf{x}_2 = 0$$

Estimating Fundamental Matrix

- Given a correspondence

$$\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$$

- The basic incidence relation is

$$\mathbf{x}_1^T \mathbf{F} \mathbf{x}_2 = 0 \quad \Leftrightarrow \quad [x_1x_2, x_1y_2, x_1, y_1x_2, y_1y_2, y_1, x_2, y_2, 1] \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

Need 8 points

Estimating Fundamental Matrix

$\mathbf{x}_1^T \mathbf{F} \mathbf{x}_2 = 0$ for 8 point correspondences:

$$\mathbf{x}_1^1 \leftrightarrow \mathbf{x}_2^1, \mathbf{x}_1^2 \leftrightarrow \mathbf{x}_2^2, \mathbf{x}_1^3 \leftrightarrow \mathbf{x}_2^3, \mathbf{x}_1^4 \leftrightarrow \mathbf{x}_2^4, \mathbf{x}_1^5 \leftrightarrow \mathbf{x}_2^5, \mathbf{x}_1^6 \leftrightarrow \mathbf{x}_2^6, \mathbf{x}_1^7 \leftrightarrow \mathbf{x}_2^7, \mathbf{x}_1^8 \leftrightarrow \mathbf{x}_2^8$$

$$\begin{bmatrix} x_1^1 x_2^1 & x_1^1 y_2^1 & x_1^1 & y_1^1 x_2^1 & y_1^1 y_2^1 & y_1^1 & x_2^1 & y_2^1 & 1 \\ x_1^2 x_2^2 & x_1^2 y_2^2 & x_1^2 & y_1^2 x_2^2 & y_1^2 y_2^2 & y_1^2 & x_2^2 & y_2^2 & 1 \\ x_1^3 x_2^3 & x_1^3 y_2^3 & x_1^3 & y_1^3 x_2^3 & y_1^3 y_2^3 & y_1^3 & x_2^3 & y_2^3 & 1 \\ x_1^4 x_2^4 & x_1^4 y_2^4 & x_1^4 & y_1^4 x_2^4 & y_1^4 y_2^4 & y_1^4 & x_2^4 & y_2^4 & 1 \\ x_1^5 x_2^5 & x_1^5 y_2^5 & x_1^5 & y_1^5 x_2^5 & y_1^5 y_2^5 & y_1^5 & x_2^5 & y_2^5 & 1 \\ x_1^6 x_2^6 & x_1^6 y_2^6 & x_1^6 & y_1^6 x_2^6 & y_1^6 y_2^6 & y_1^6 & x_2^6 & y_2^6 & 1 \\ x_1^7 x_2^7 & x_1^7 y_2^7 & x_1^7 & y_1^7 x_2^7 & y_1^7 y_2^7 & y_1^7 & x_2^7 & y_2^7 & 1 \\ x_1^8 x_2^8 & x_1^8 y_2^8 & x_1^8 & y_1^8 x_2^8 & y_1^8 y_2^8 & y_1^8 & x_2^8 & y_2^8 & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{Af} = \mathbf{0}$$

Direct Linear Transformation (DLT)

Algebraic Error vs. Geometric Error

- Algebraic Error

$$\min \|\mathbf{A}\mathbf{f}\|$$

- Geometric Error (better) Unit: pixel

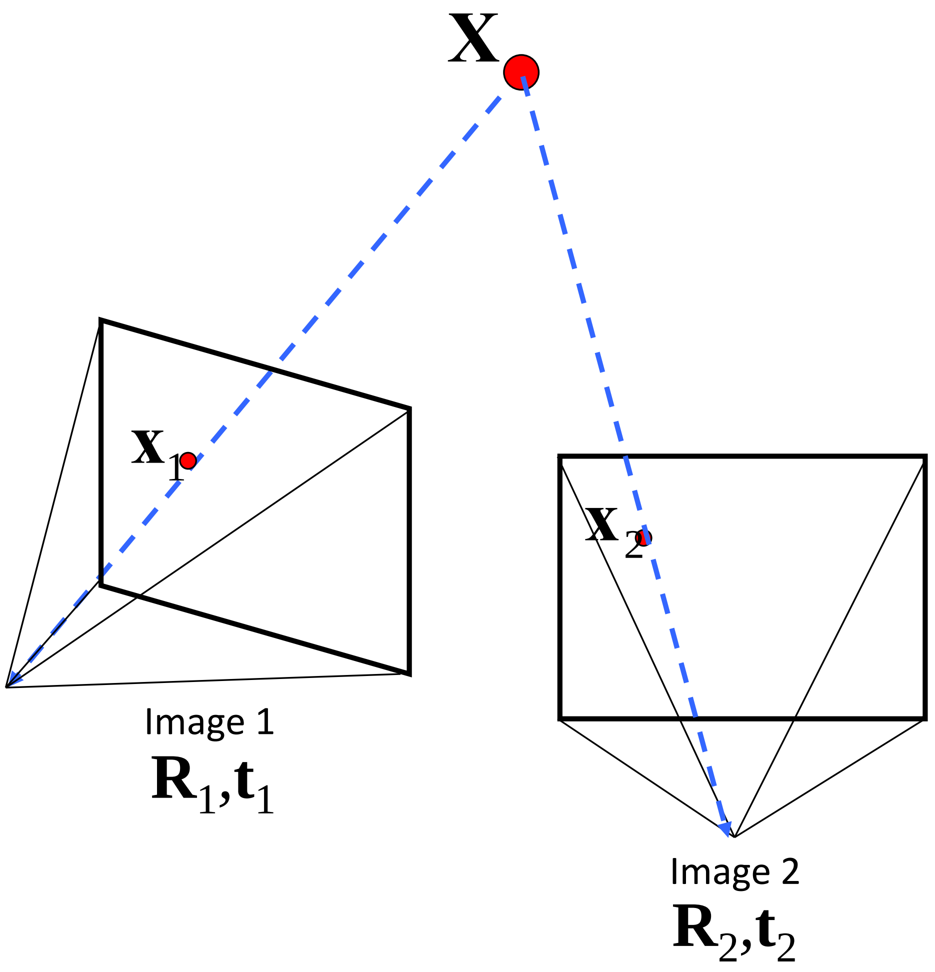
$$\min \sum_{j} d\left(\mathbf{x}_1^j, \mathbf{F}\mathbf{x}_2^j\right)^2 + d\left(\mathbf{x}_2^j, \mathbf{F}^T\mathbf{x}_1^j\right)^2$$

Solved by (non-linear) least square solver (e.g. Ceres)

RANSAC to Estimate Fundamental Matrix

- For many times
  - Pick 8 points
  - Compute a solution for $\mathbf{F}$ using these 8 points
  - Count number of inliers
- Pick the one with the largest number of inliers

Fundamental Matrix → Essential Matrix



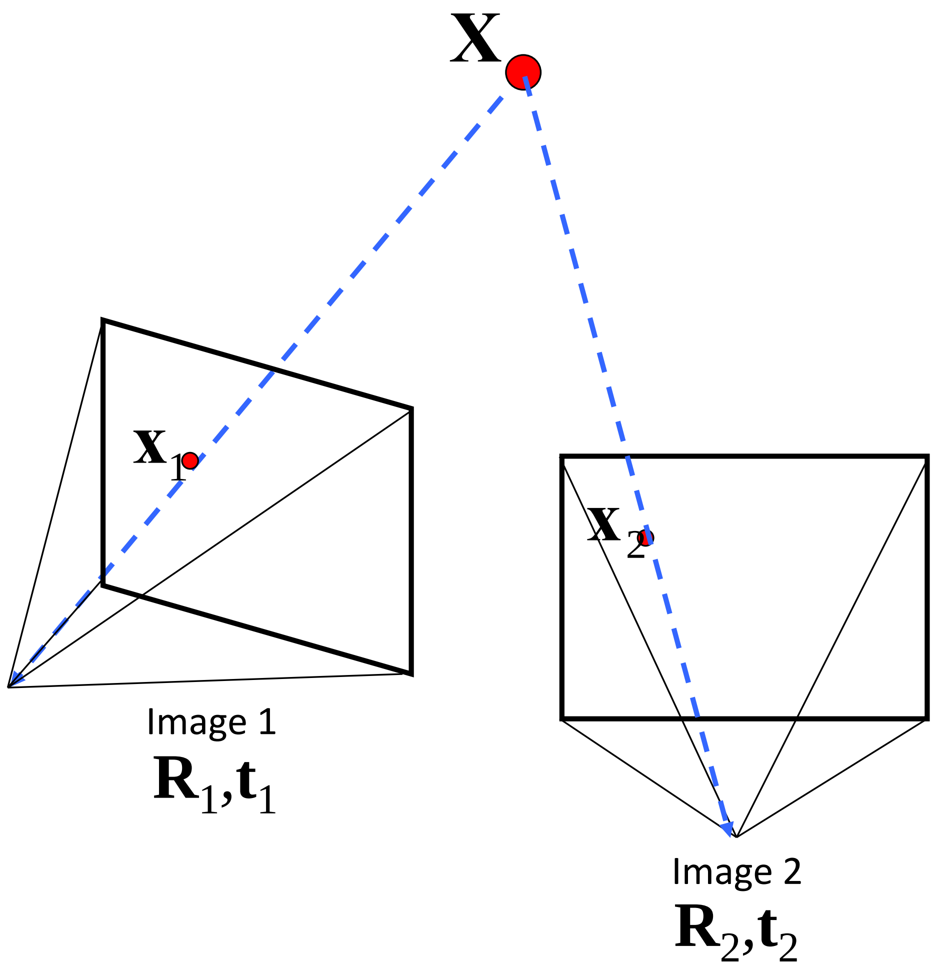
$$\mathbf{x}_1^T \mathbf{F} \mathbf{x}_2 = 0$$

$$\mathbf{E} = \mathbf{K}_1^T \mathbf{F} \mathbf{K}_2$$

$$\mathbf{x}_1 = \mathbf{K}\begin{bmatrix}\mathbf{R}_1 | \mathbf{t}_1\end{bmatrix}\mathbf{X}$$

$$\mathbf{x}_2 = \mathbf{K}\begin{bmatrix}\mathbf{R}_2 | \mathbf{t}_2\end{bmatrix}\mathbf{X}$$

Essential Matrix → $\begin{bmatrix}\mathbf{R} | \mathbf{t}\end{bmatrix}$



$$\mathbf{x}_1^T \mathbf{F} \mathbf{x}_2 = 0$$

$$\mathbf{E} = \mathbf{K}_1^T \mathbf{F} \mathbf{K}_2$$

Essential Matrix → $\left[\mathbf{R} \middle| \mathbf{t}\right]$

Result 9.19. For a given essential matrix

$$\mathbf{E} = \mathbf{U}\mathrm{diag}\left(1,1,0\right)\mathbf{V}^T,$$

and the first camera matrix $\mathbf{P}_1 = \left[\mathbf{I} \middle| \mathbf{0}\right]$, there are four possible choices for the second camera matrix $\mathbf{P}_2$:

$$\mathbf{P}_2 = \left[\mathbf{U}\mathbf{W}\mathbf{V}^T \middle| +\mathbf{u}_3\right]$$
$$\mathbf{P}_2 = \left[\mathbf{U}\mathbf{W}\mathbf{V}^T \middle| -\mathbf{u}_3\right]$$
$$\mathbf{P}_2 = \left[\mathbf{U}\mathbf{W}^T\mathbf{V}^T \middle| +\mathbf{u}_3\right]$$
$$\mathbf{P}_2 = \left[\mathbf{U}\mathbf{W}^T\mathbf{V}^T \middle| -\mathbf{u}_3\right]$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Page 259 of the bible (Multiple View Geometry, 2nd Ed)

Four Possible Solutions

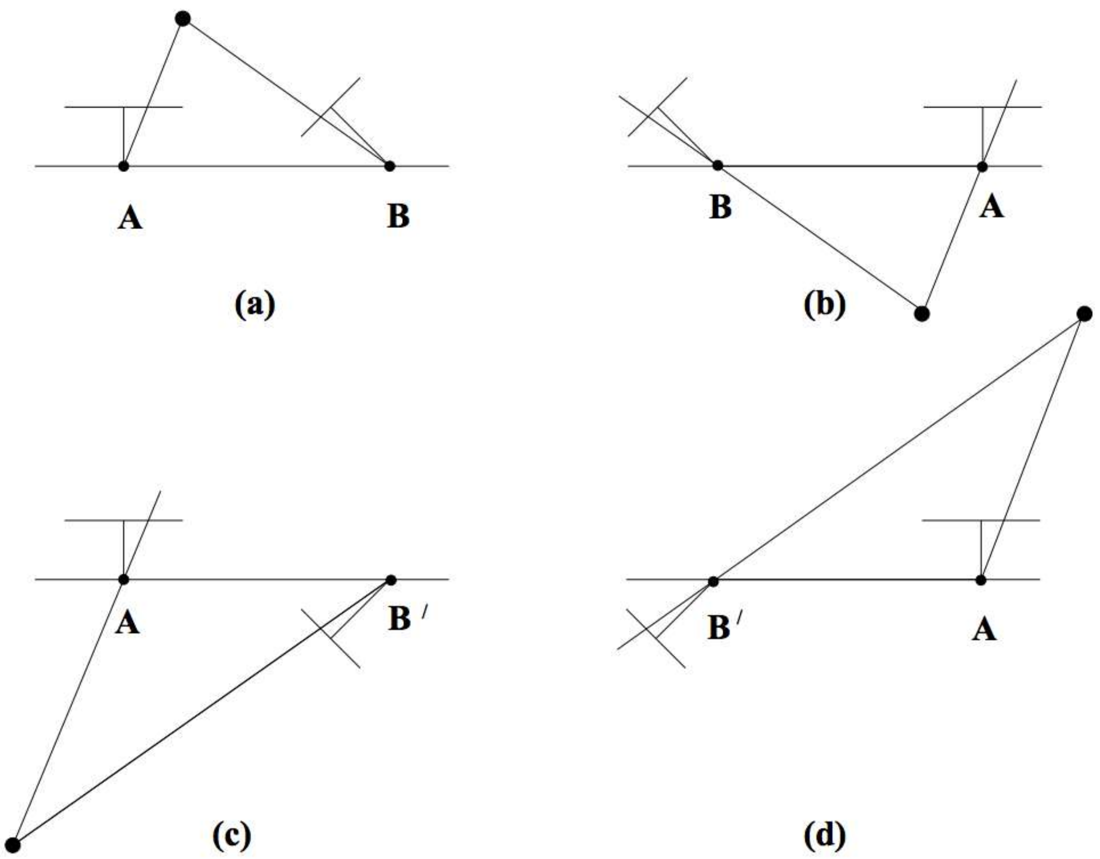


Fig. 9.12. **The four possible solutions for calibrated reconstruction from E.** *Between the left and right sides there is a baseline reversal. Between the top and bottom rows camera B rotates* 180° *about the baseline. Note, only in (a) is the reconstructed point in front of both cameras.*

Page 260 of the bible (Multiple View Geometry, 2nd Ed)

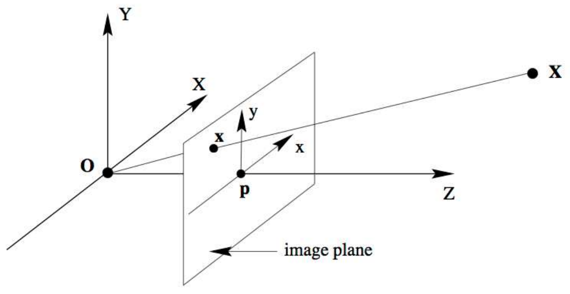
In front of the camera?

- Camera Extrinsic $\left[\mathbf{R}|\mathbf{t}\right]$

$$\begin{bmatrix}X_{cam}\\Y_{cam}\\Z_{cam}\end{bmatrix}=\mathbf{R}\begin{bmatrix}X_{world}\\Y_{world}\\Z_{world}\end{bmatrix}+\mathbf{t} \Leftrightarrow \begin{bmatrix}X_{world}\\Y_{world}\\Z_{world}\end{bmatrix}=\mathbf{R}^{-1}\left(\begin{bmatrix}X_{cam}\\Y_{cam}\\Z_{cam}\end{bmatrix}-\mathbf{t}\right)=\mathbf{R}^{T}\begin{bmatrix}X_{cam}\\Y_{cam}\\Z_{cam}\end{bmatrix}-\mathbf{R}^{T}\mathbf{t}$$

- Camera Center

$$\begin{bmatrix}X_{cam}\\Y_{cam}\\Z_{cam}\end{bmatrix}=\begin{bmatrix}0\\0\\0\end{bmatrix} \Rightarrow \mathbf{C}=\begin{bmatrix}X_{world}\\Y_{world}\\Z_{world}\end{bmatrix}=\mathbf{R}^{T}\begin{bmatrix}0\\0\\0\end{bmatrix}-\mathbf{R}^{T}\mathbf{t}=-\mathbf{R}^{T}\mathbf{t}$$



- View Direction

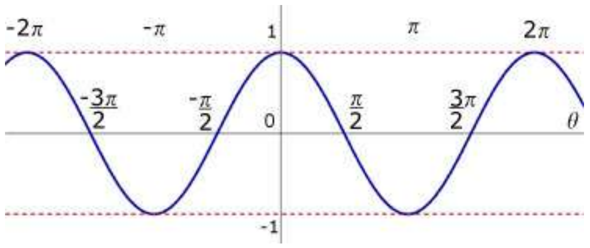
$$\begin{bmatrix}0\\0\\1\end{bmatrix}-\begin{bmatrix}0\\0\\0\end{bmatrix} \Rightarrow \left(\mathbf{R}^{T}\begin{bmatrix}0\\0\\1\end{bmatrix}-\mathbf{R}^{T}\mathbf{t}\right)-\left(\mathbf{C}\right)=\left(\mathbf{R}(3,:)^{T}-\mathbf{R}^{T}\mathbf{t}\right)-\left(-\mathbf{R}^{T}\mathbf{t}\right)=\mathbf{R}(3,:)^{T}$$

Camera Coordinate System World Coordinate System

In front of the camera?

- A point $\mathbf{X}$
- Direction from camera center to point $\mathbf{X} - \mathbf{C}$
- Angle Between Two Vectors

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\|\|\mathbf{B}\|\cos\theta$$



- Angle Between $\mathbf{X} - \mathbf{C}$ and View Direction
- Just need to test

$$\left(\mathbf{X} - \mathbf{C}\right) \cdot \mathbf{R}(3,:)^T > 0?$$

Pick the Solution

With maximal number of points in front of both cameras.

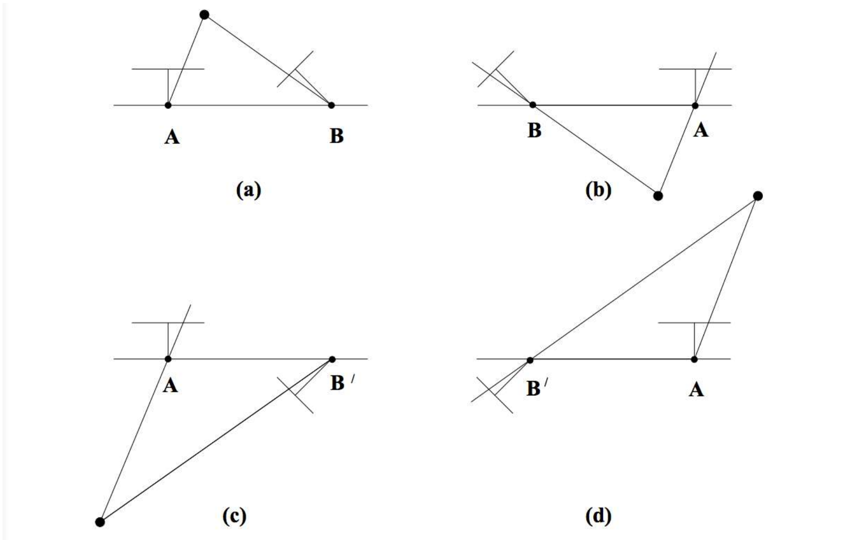
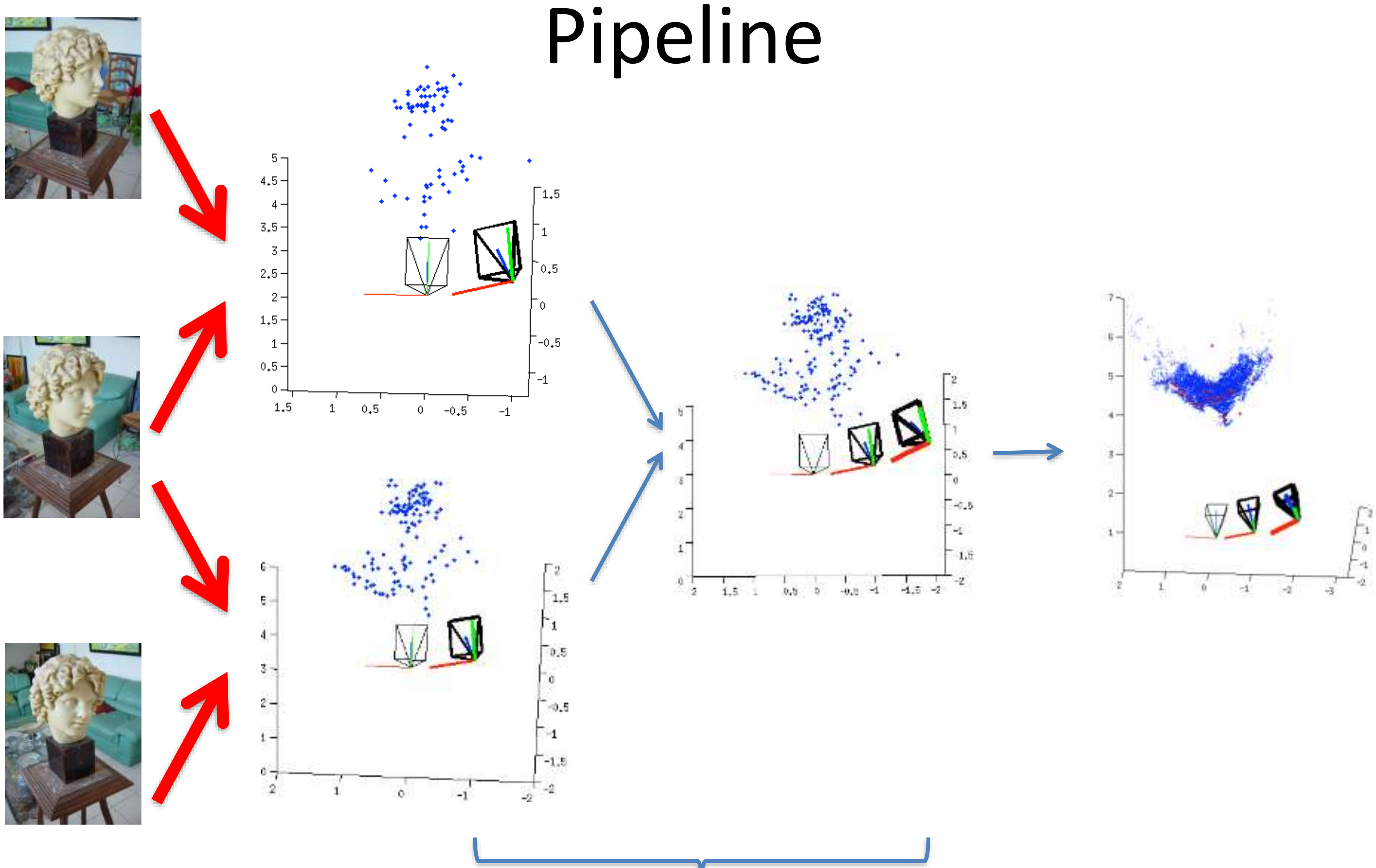


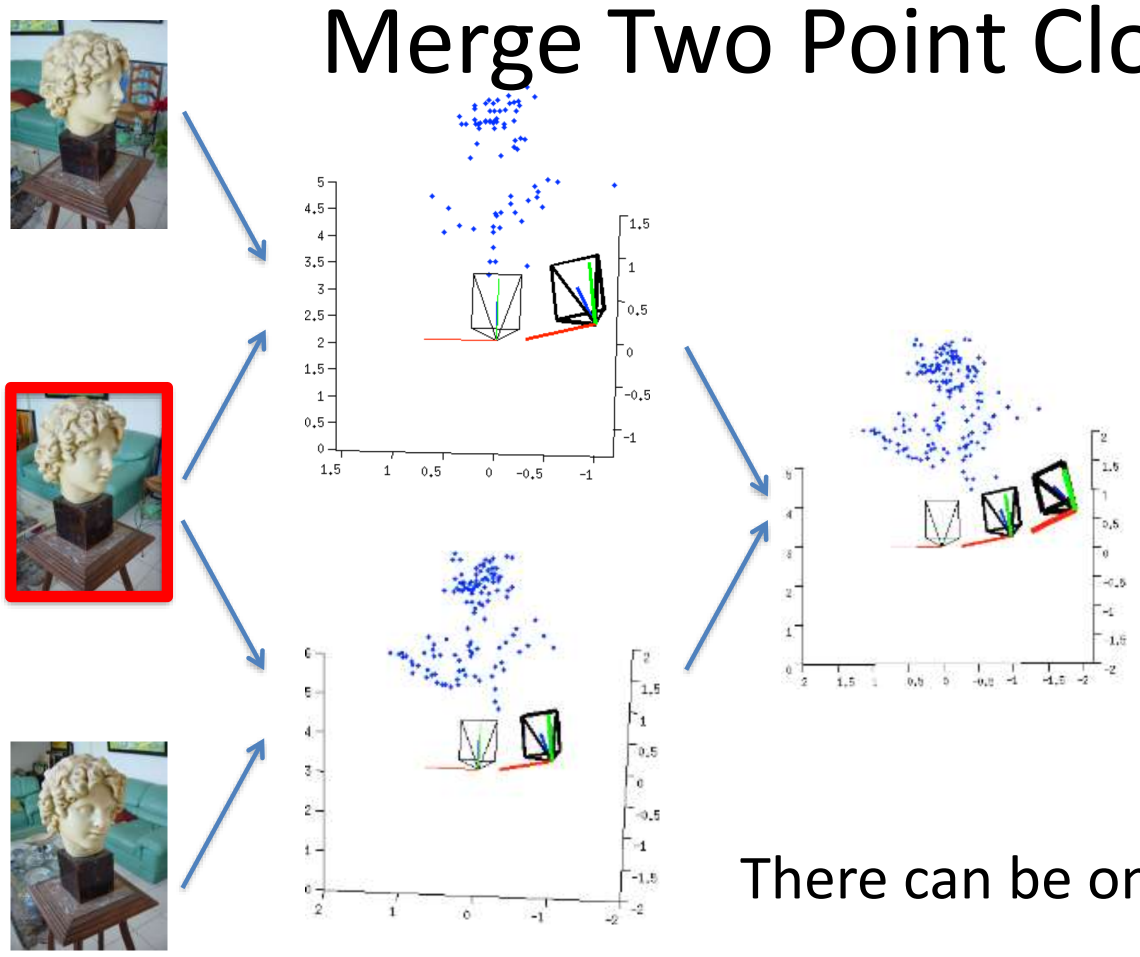
Fig. 9.12. **The four possible solutions for calibrated reconstruction from E.** *Between the left and right sides there is a baseline reversal. Between the top and bottom rows camera B rotates* 180° *about the baseline. Note, only in (a) is the reconstructed point in front of both cameras.*

Page 260 of the bible (Multiple View Geometry, 2nd Ed)

Pipeline

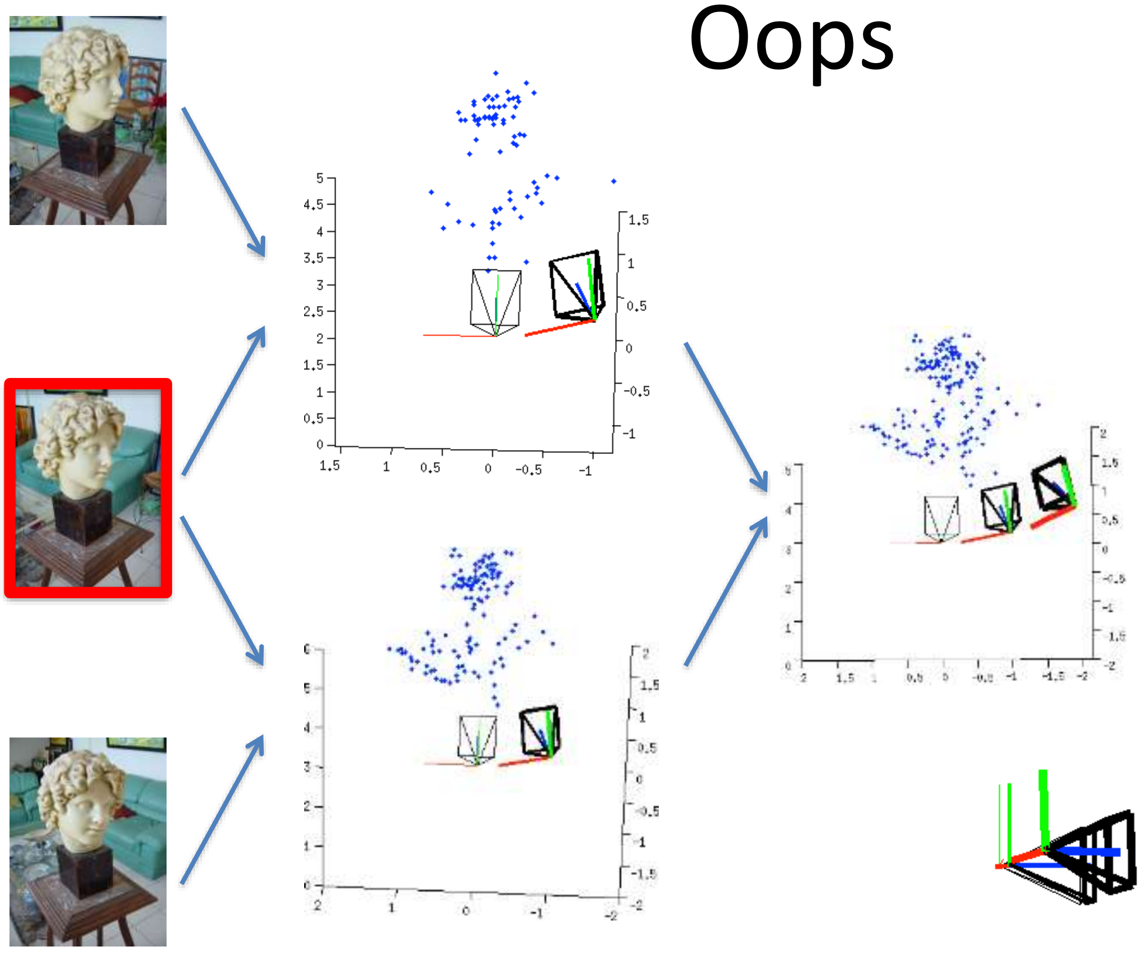


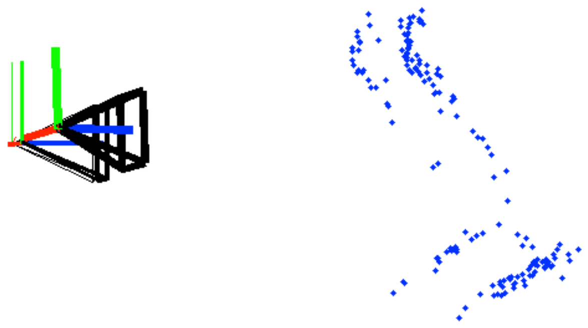
Merge Two Point Cloud



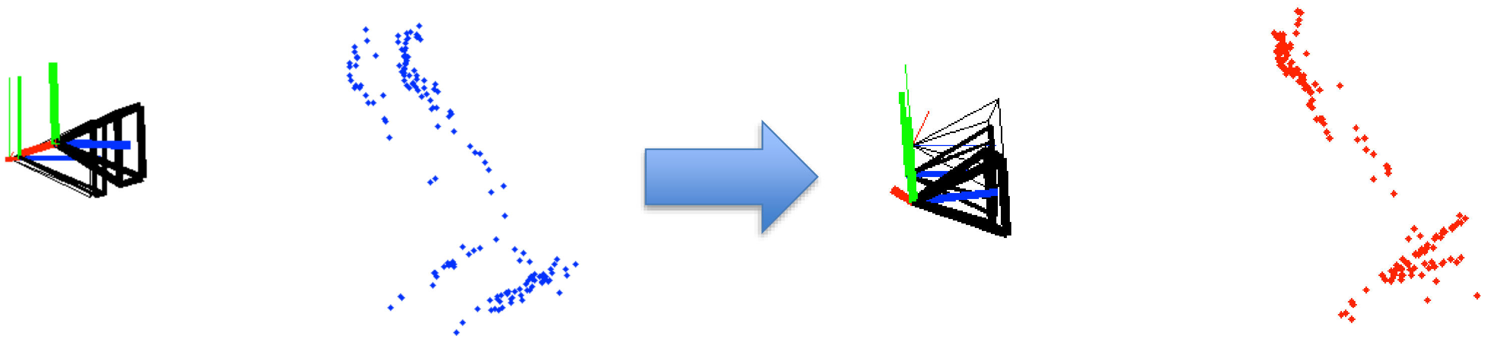
There can be only one $\left[\mathbf{R}_2 | \mathbf{t}_2\right]$

Oops




See From a Different Angle

Bundle Adjustment



Rethinking the SFM problem

- Input: Observed 2D image position

$$\begin{array}{ccc} \tilde{\mathbf{x}}_1^1 & \tilde{\mathbf{x}}_1^2 & \\ \tilde{\mathbf{x}}_2^1 & \tilde{\mathbf{x}}_2^2 & \tilde{\mathbf{x}}_2^3 \\ \tilde{\mathbf{x}}_3^1 & & \tilde{\mathbf{x}}_3^3 \end{array}$$

- Output:

  Unknown Camera Parameters (with some guess)

$$\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right], \left[\mathbf{R}_2 \middle| \mathbf{t}_2\right], \left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]$$

  Unknown Point 3D coordinate (with some guess)

$$\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3, \cdots$$

Bundle Adjustment

A valid solution $\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right], \left[\mathbf{R}_2 \middle| \mathbf{t}_2\right], \left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]$ and $\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3, \cdots$ must let

$$\text{Re-projection} \left\{ \begin{array}{lll} \mathbf{x}_1^1 = \mathbf{K}\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right]\mathbf{X}^1 & \mathbf{x}_1^2 = \mathbf{K}\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right]\mathbf{X}^2 & \\ \mathbf{x}_2^1 = \mathbf{K}\left[\mathbf{R}_2 \middle| \mathbf{t}_2\right]\mathbf{X}^1 & \mathbf{x}_2^2 = \mathbf{K}\left[\mathbf{R}_2 \middle| \mathbf{t}_2\right]\mathbf{X}^2 & \mathbf{x}_2^3 = \mathbf{K}\left[\mathbf{R}_2 \middle| \mathbf{t}_2\right]\mathbf{X}^3 \\ \mathbf{x}_3^1 = \mathbf{K}\left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]\mathbf{X}^1 & & \mathbf{x}_3^3 = \mathbf{K}\left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]\mathbf{X}^3 \end{array} \right.$$

$$=$$

$$\text{Observation} \left\{ \begin{array}{lll} \tilde{\mathbf{x}}_1^1 & \tilde{\mathbf{x}}_1^2 & \\ \tilde{\mathbf{x}}_2^1 & \tilde{\mathbf{x}}_2^2 & \tilde{\mathbf{x}}_2^3 \\ \tilde{\mathbf{x}}_3^1 & & \tilde{\mathbf{x}}_3^3 \end{array} \right.$$

Bundle Adjustment

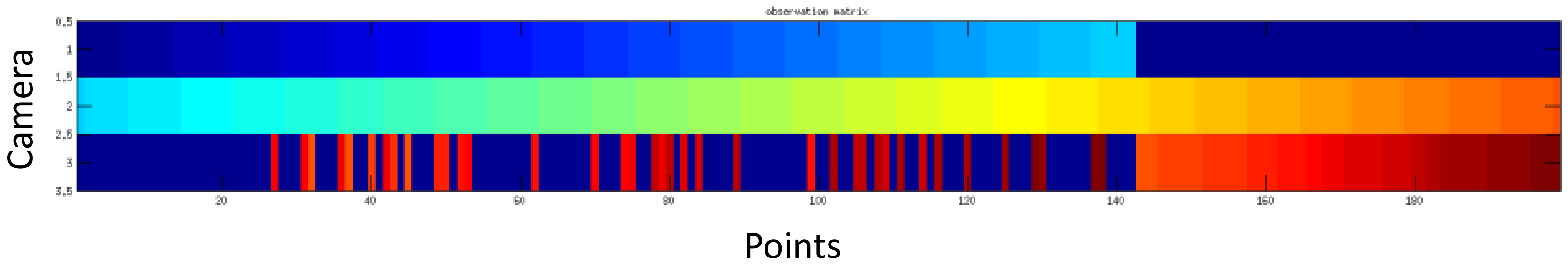
A valid solution $\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right], \left[\mathbf{R}_2 \middle| \mathbf{t}_2\right], \left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]$ and $\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3, \cdots$ must let the Re-projection close to the Observation, i.e. to minimize the reprojection error

$$\min \sum_i \sum_j \left(\tilde{\mathbf{x}}_i^j - \mathbf{K}\left[\mathbf{R}_i \middle| \mathbf{t}_i\right]\mathbf{X}^j \right)^2$$

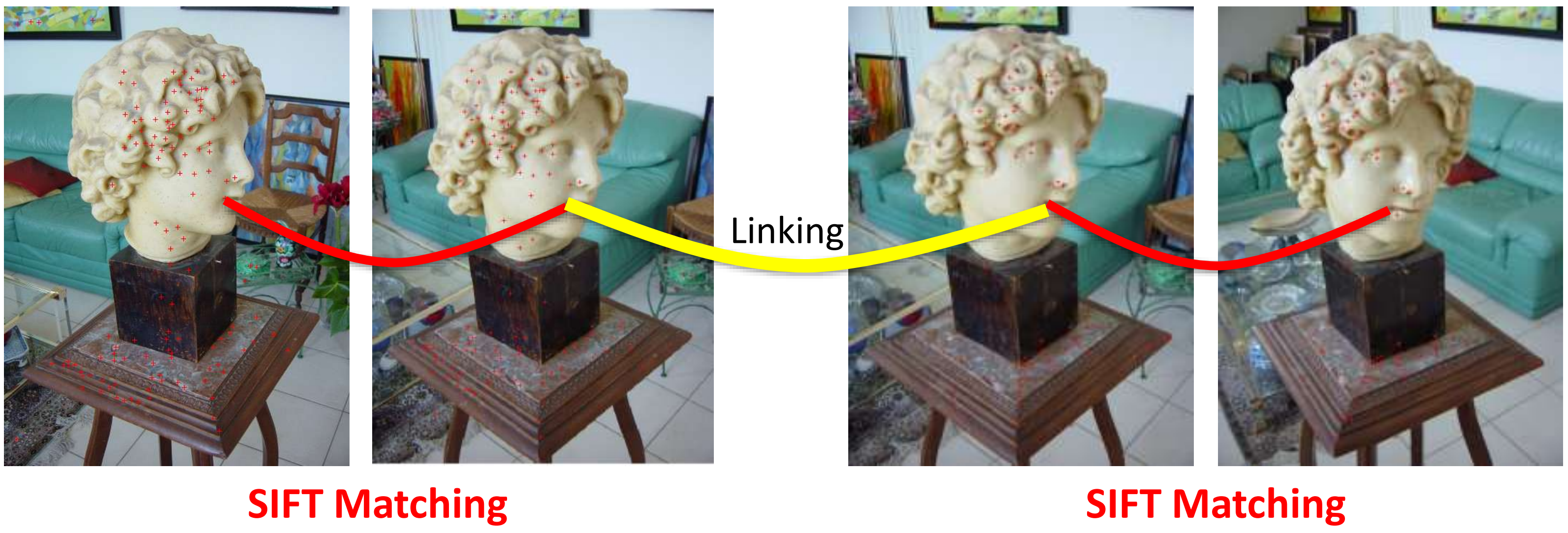
Bundle Adjustment

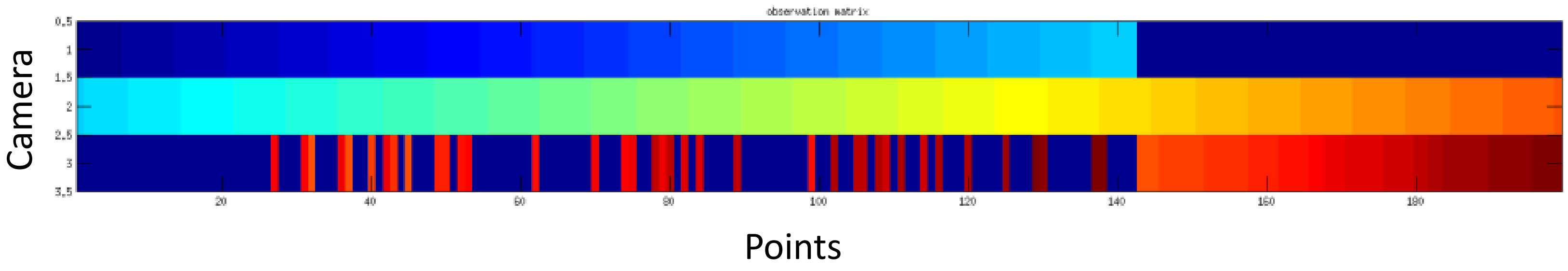
A valid solution $\left[\mathbf{R}_1|\mathbf{t}_1\right],\left[\mathbf{R}_2|\mathbf{t}_2\right],\left[\mathbf{R}_3|\mathbf{t}_3\right]$ and $\mathbf{X}^1,\mathbf{X}^2,\mathbf{X}^3,\cdots$ must let the Re-projection close to the Observation, i.e. to minimize the reprojection error

$$\min \sum_i \sum_j \left(\tilde{\mathbf{x}}_i^j - \mathbf{K}\left[\mathbf{R}_i|\mathbf{t}_i\right]\mathbf{X}^j\right)^2$$



Linking


Linking
SIFT Matching
SIFT Matching


observation matrix
Camera
Points

Solving This Optimization Problem

- Theory:

  The Levenberg–Marquardt algorithm

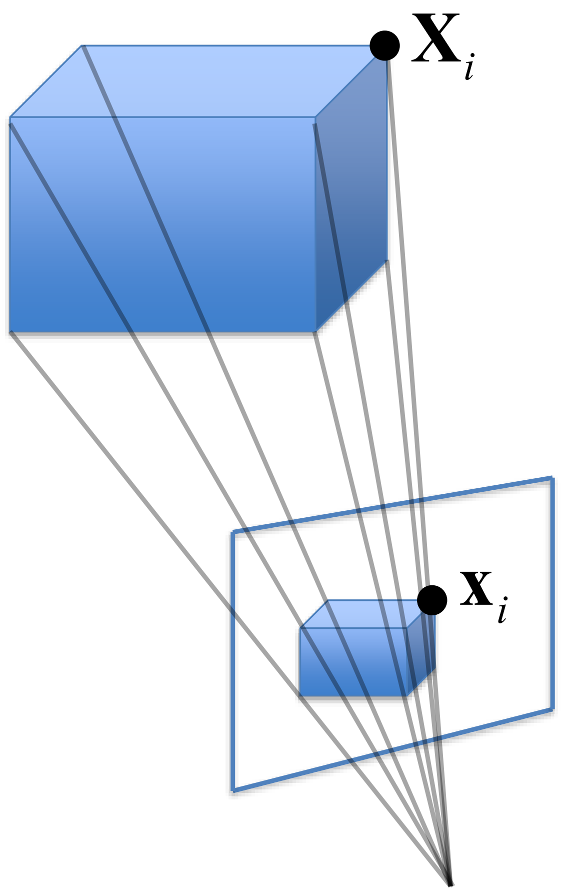
  http://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm

- Practice:

  The Ceres-Solver from Google

  http://code.google.com/p/ceres-solver/

Non-linear Least Square Cuboid Reconstruction



Observation:

Pixel location of the 7 (or 6) corners

Parameters to be estimated:

Cuboid: Height h x Width w x 1

Camera: Intrinsic K and Extrinsic R, t

$$\min \sum_i \left\| \mathbf{x}_i - \mathbf{K}\left(\mathbf{R}\mathbf{X}_i + \mathbf{t}\right) \right\|^2$$

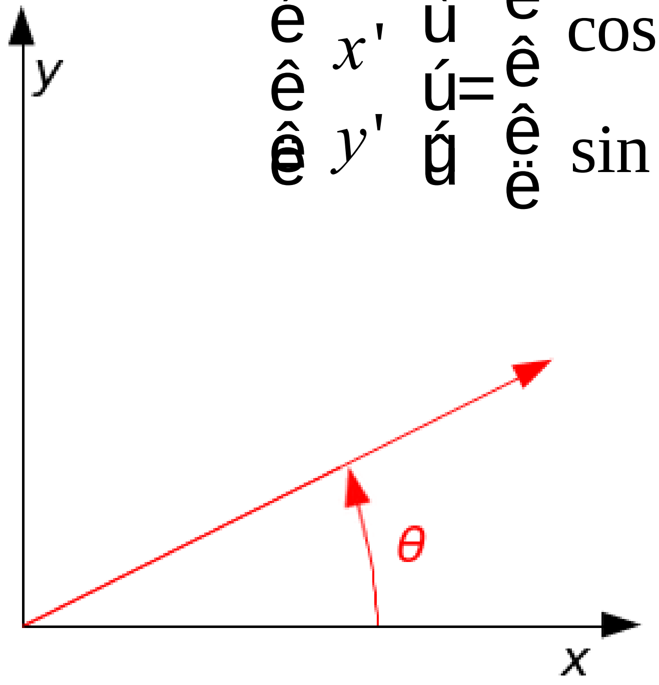
$$\mathbf{X} = \begin{bmatrix} h & h & h & h & -h & -h & -h & -h \\ w & w & -w & -w & w & w & -w & -w \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

http://mit.edu/jxiao/Public/software/fitCuboid/

Parameterizing Rotation Matrix

- 2D Rotation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(q) & -\sin(q) \\ \sin(q) & \cos(q) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



$$\mathbf{R}^T = \mathbf{R}^{-1}, \det \mathbf{R} = 1$$

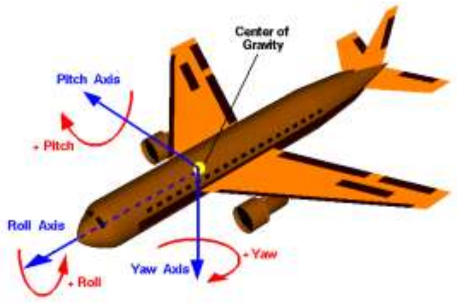
3D Rotation

$$\mathbf{R}_x(q) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(q) & -\sin(q) \\ 0 & \sin(q) & \cos(q) \end{bmatrix}$$

$$\mathbf{R}_y(q) = \begin{bmatrix} \cos(q) & 0 & \sin(q) \\ 0 & 1 & 0 \\ -\sin(q) & 0 & \cos(q) \end{bmatrix}$$

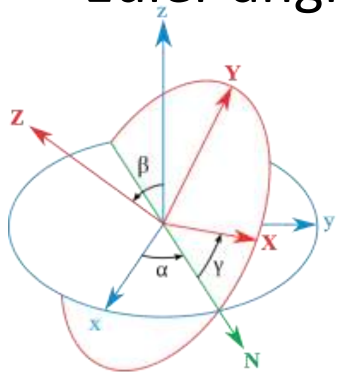
$$\mathbf{R}_z(q) = \begin{bmatrix} \cos(q) & -\sin(q) & 0 \\ \sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Yaw, pitch and roll are a, b, g



$$\mathbf{R}_x(g)\mathbf{R}_y(b)\mathbf{R}_z(a)$$

Euler angles are a, b, g

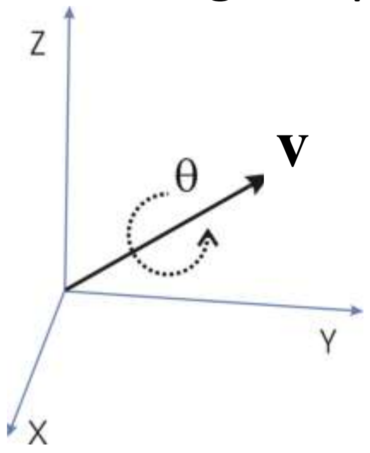


$$\mathbf{R}_z(g)\mathbf{R}_x(b)\mathbf{R}_y(a)$$

http://en.wikipedia.org/wiki/Rotation_matrix

3D Rotation

Axis-angle representation



Quaternions

$$\mathbf{q} = \left(\mathbf{v}\sin\left(\frac{\theta}{2}\right), \cos\left(\frac{\theta}{2}\right)\right)^T$$

Avoid Gimbal Lock!

Recommendation!

Triplet Representation $\mathbf{v}\theta$ (3 dof) ← Not over-parameterized

Rodrigues' rotation formula

$$\mathbf{k}_{rot} = \mathbf{k}\cos\theta + (\mathbf{v} \times \mathbf{k})\sin\theta + \mathbf{v}(\mathbf{v} \cdot \mathbf{k})(1 - \cos\theta)$$

http://en.wikipedia.org/wiki/Axis-angle_representation
http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
http://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

Matlab Symbolic Operation

How to derive complicated equations (after passing math classes)?

```
syms  h, w, K, alpha, beta, gamma, tx, ty, tz
X = [h; w; f];
Y = […] * X;
Z = K* Y;
x = Z(1:2,:) ./ Z([3,3],:);
ccode(x,’file’,’x.cpp’); % ←matlab will generate C code for x
```

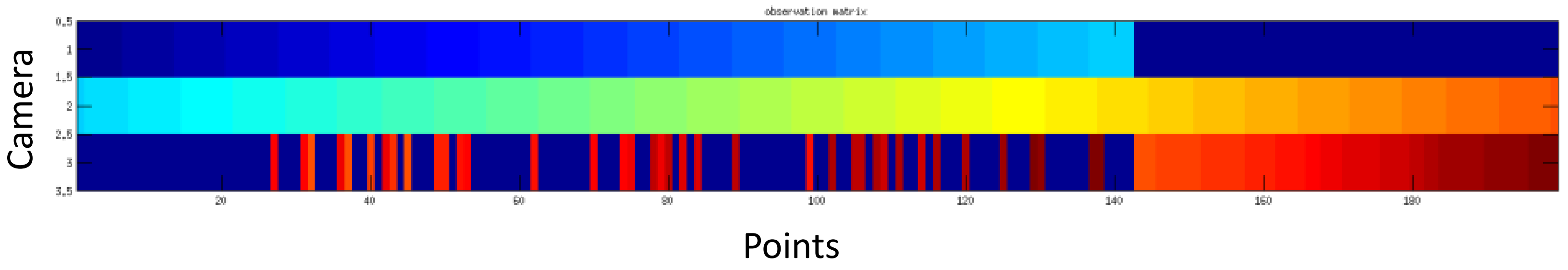
http://www.mathworks.com/help/symbolic/sym.html

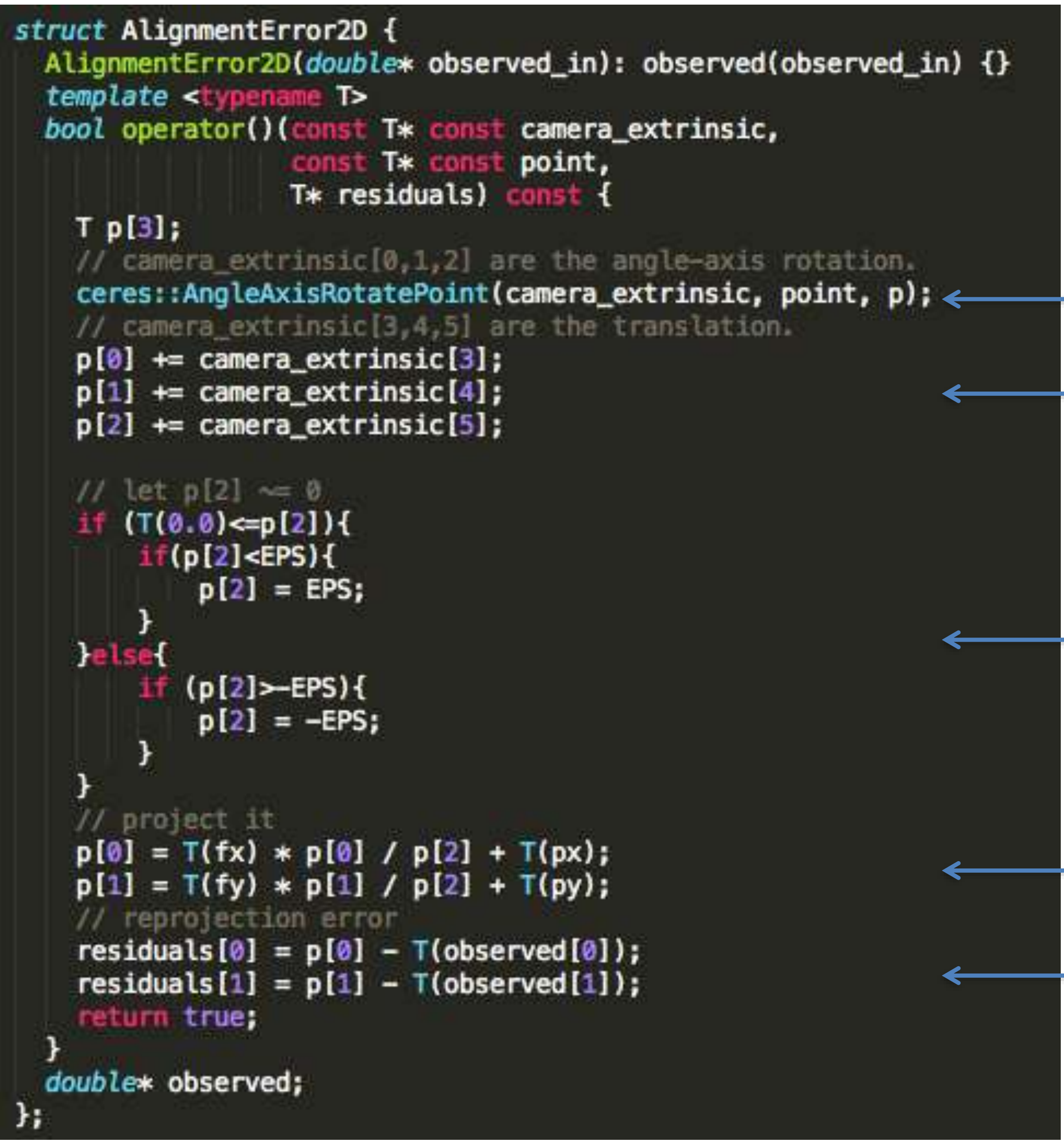
http://mit.edu/jxiao/Public/software/fitCuboid/fitCuboid/derive4cuboid.m

Ceres for Bundle Adjustment

A valid solution $\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right], \left[\mathbf{R}_2 \middle| \mathbf{t}_2\right], \left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]$ and $\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3, \cdots$ must let the Re-projection close to the Observation, i.e. to minimize the reprojection error

$$\min \sum_i \sum_j \left(\tilde{\mathbf{x}}_i^j - \mathbf{K}\left[\mathbf{R}_i \middle| \mathbf{t}_i\right] \mathbf{X}^j \right)^2$$





```
struct AlignmentError2D {
  AlignmentError2D(double* observed_in): observed(observed_in) {}
  template <typename T>
  bool operator()(const T* const camera_extrinsic,
                  const T* const point,
                  T* residuals) const {
    T p[3];
    // camera_extrinsic[0,1,2] are the angle-axis rotation.
    ceres::AngleAxisRotatePoint(camera_extrinsic, point, p);
    // camera_extrinsic[3,4,5] are the translation.
    p[0] += camera_extrinsic[3];
    p[1] += camera_extrinsic[4];
    p[2] += camera_extrinsic[5];

    // let p[2] ~= 0
    if (T(0.0)<=p[2]){
        if(p[2]<EPS){
            p[2] = EPS;
        }
    }else{
        if (p[2]>-EPS){
            p[2] = -EPS;
        }
    }
    // project it
    p[0] = T(fx) * p[0] / p[2] + T(px);
    p[1] = T(fy) * p[1] / p[2] + T(py);
    // reprojection error
    residuals[0] = p[0] - T(observed[0]);
    residuals[1] = p[1] - T(observed[1]);
    return true;
  }
  double* observed;
};
```

ba2D.cc

$\mathbf{RX}$

$\mathbf{RX}+\mathbf{t}=\left[\mathbf{R}\,|\,\mathbf{t}\right]\mathbf{X}$

To avoid divide by 0

$\mathbf{x}=\mathbf{K}\left[\mathbf{R}\,|\,\mathbf{t}\right]\mathbf{X}$

$\mathbf{x}-\tilde{\mathbf{x}}$

Initialization Matters

- Input: Observed 2D image position

$$\begin{array}{ccc} \tilde{\mathbf{x}}_1^1 & \tilde{\mathbf{x}}_1^2 & \\ \tilde{\mathbf{x}}_2^1 & \tilde{\mathbf{x}}_2^2 & \tilde{\mathbf{x}}_2^3 \\ \tilde{\mathbf{x}}_3^1 & & \tilde{\mathbf{x}}_3^3 \end{array}$$

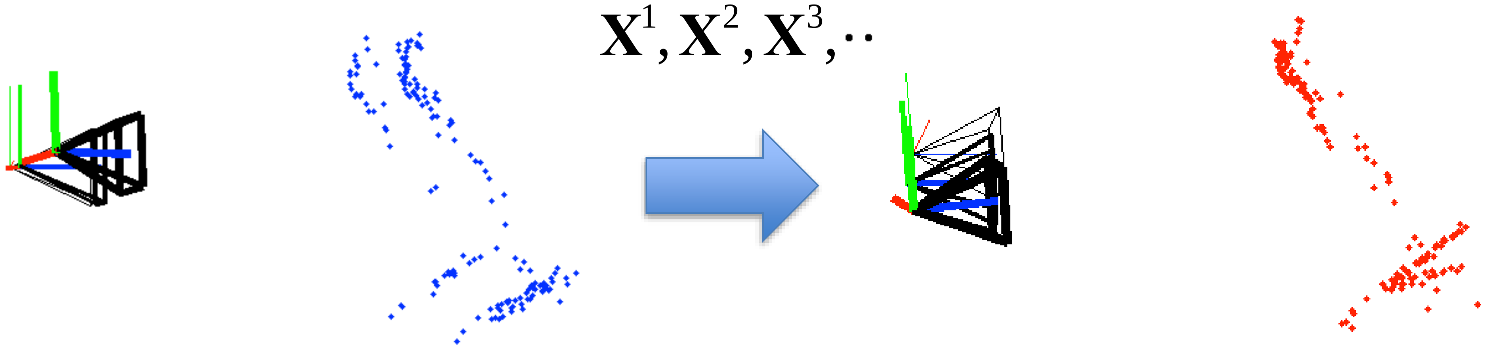
- Output:

Unknown Camera Parameters (with some guess)

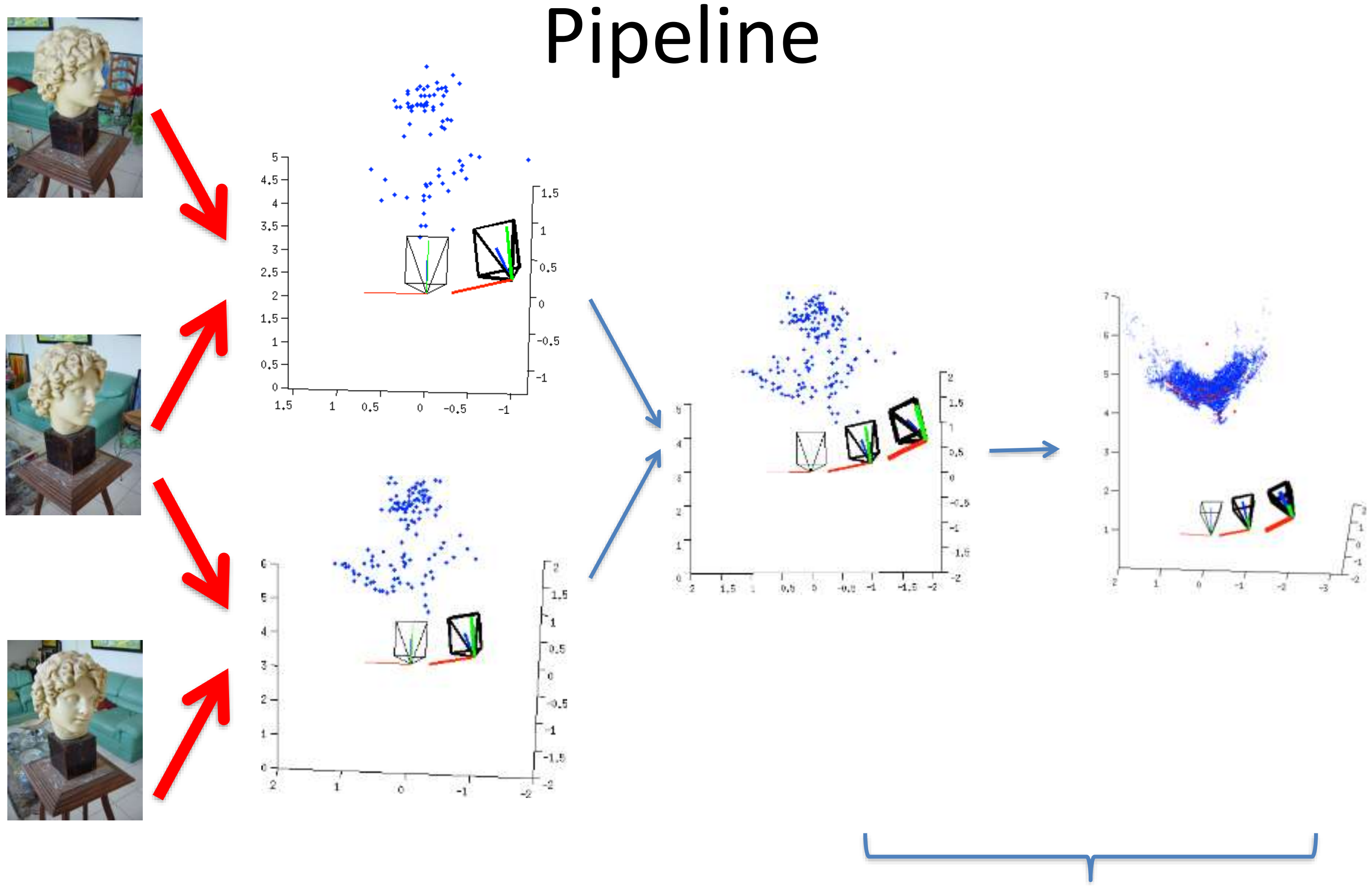
$$\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right], \left[\mathbf{R}_2 \middle| \mathbf{t}_2\right], \left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]$$

Unknown Point 3D coordinate (with some guess)

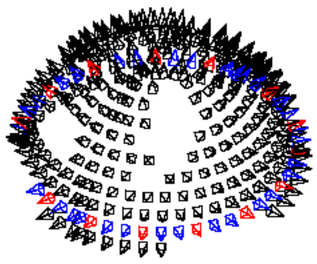
$$\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3, \cdots$$



Pipeline



Multiple View Stereo



State-of-the-art:

PMVS: http://grail.cs.washington.edu/software/pmvs/

Accurate, Dense, and Robust Multi-View Stereopsis, Y Furukawa and J Ponce, 2007.

Benchmark:

http://vision.middlebury.edu/mview/

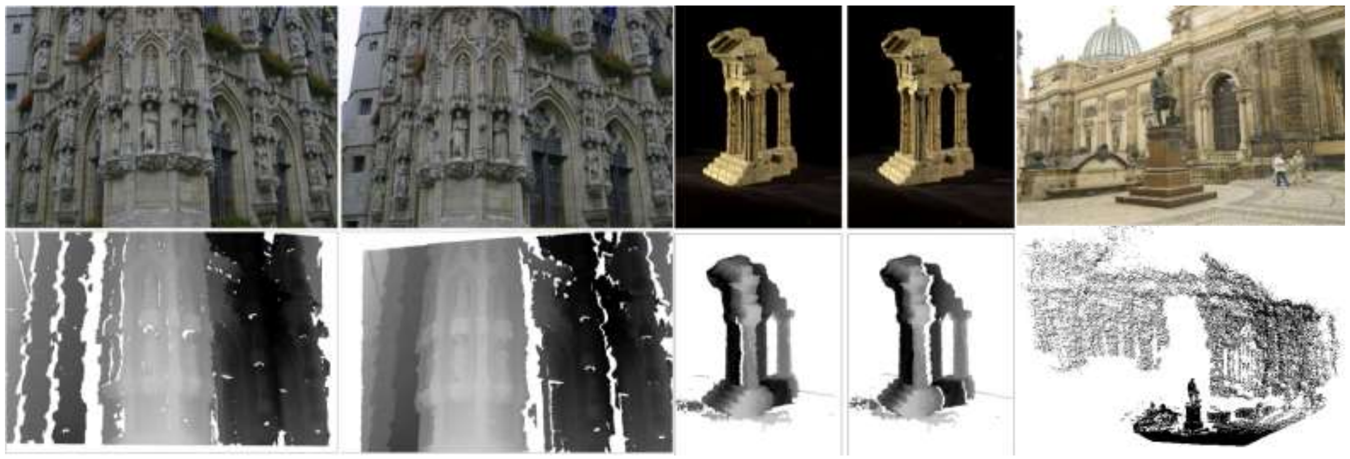
A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms.

SM Seitz, B Curless, J Diebel, D Scharstein, R Szeliski. 2006.

Baseline:

Multi-view stereo revisited. M Goesele, B Curless, SM Seitz. 2006.

Key idea: Matching Propagation



[1] Learning Two-view Stereo Matching, J Xiao, J Chen, DY Yeung, and L Quan, 2008.

[2] Accurate, Dense, and Robust Multi-View Stereopsis, Y Furukawa and J Ponce, 2007.

[3] Multi-view stereo revisited. M Goesele, B Curless, SM Seitz. 2006.

[4] Robust Dense Matching Using Local and Global Geometric Constraints, M Lhuillier & L Quan, 2000.

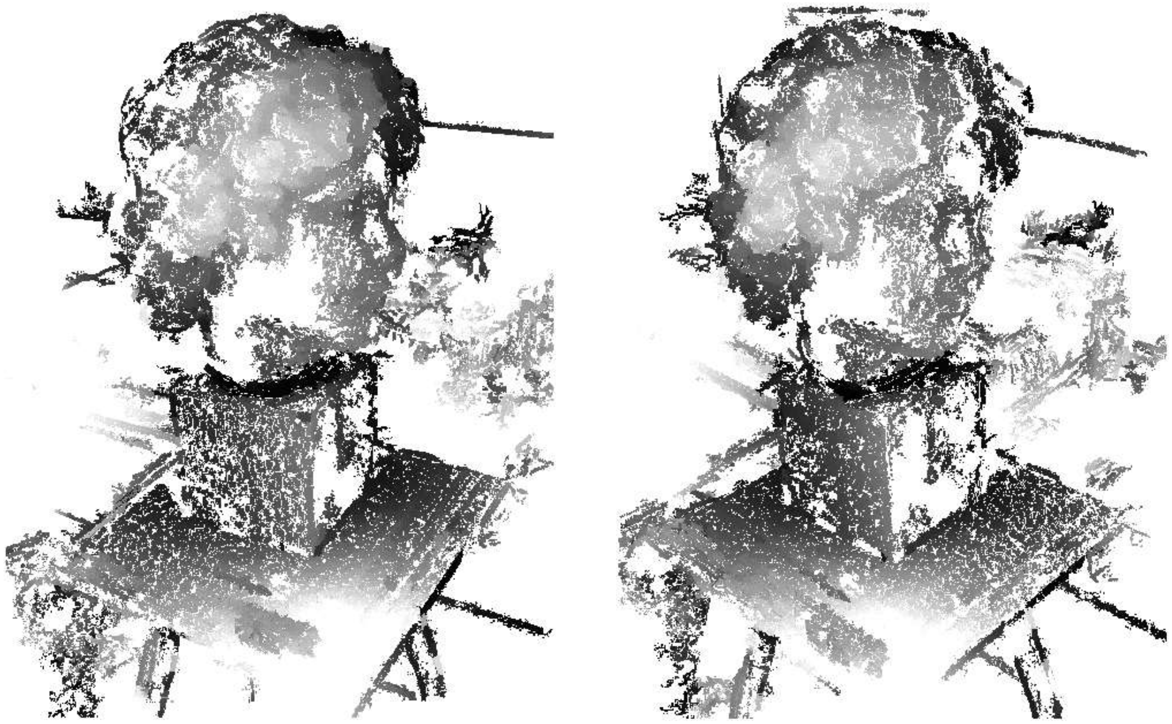
In another context:

[5] PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing,

C Barnes, E Shechtman, A Finkelstein, and DB Goldman, 2009.

Simplest Matching Propagation

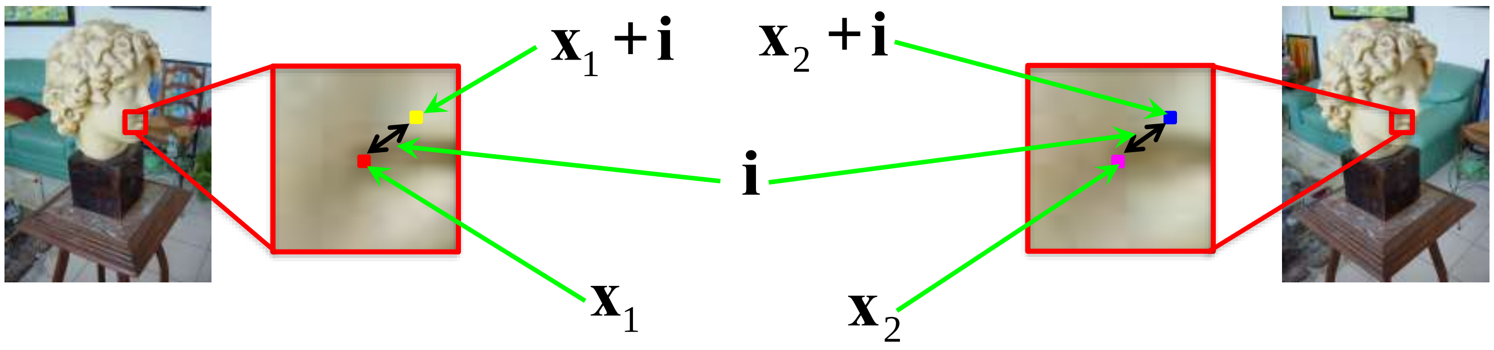
We are going to learn a very simple algorithm that is implemented in the SFMedu program.



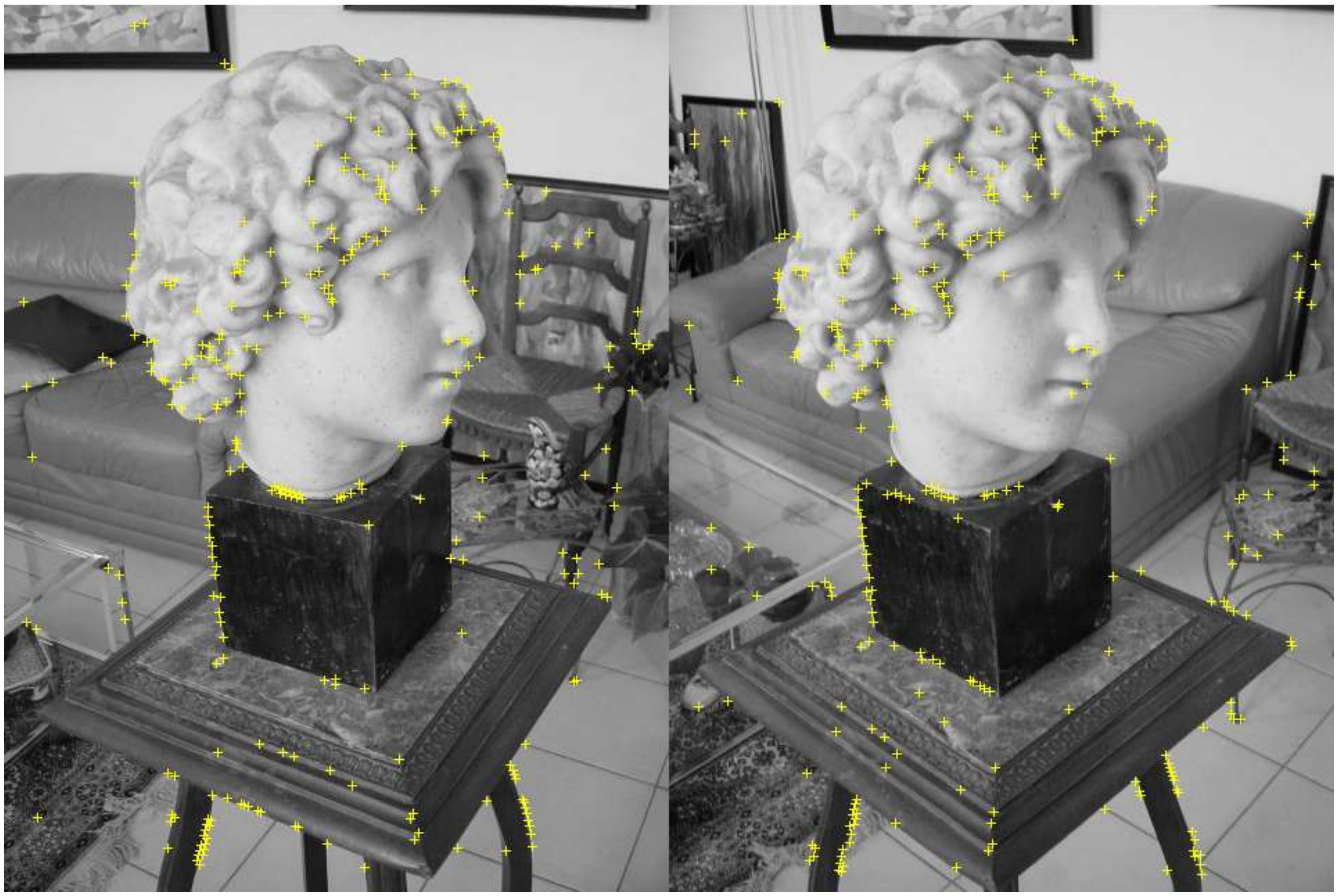
Descriptor: ZNCC (Zero-mean Normalized Cross-Correlation)

- Invariant to linear radiometric changes
- More conservative than others such as sum of absolute or square differences in uniform regions
- More tolerant in textured areas where noise might be important

$$ZNCC\left(\mathbf{x}_1, \mathbf{x}_2\right) = \frac{\sum_{\mathbf{i}} \left(I(\mathbf{x}_1 + \mathbf{i}) - \bar{I}(\mathbf{x}_1)\right)\left(I(\mathbf{x}_2 + \mathbf{i}) - \bar{I}(\mathbf{x}_2)\right)}{\sqrt{\sum_{\mathbf{i}} \left(I(\mathbf{x}_1 + \mathbf{i}) - \bar{I}(\mathbf{x}_1)\right)^2 \sum_{\mathbf{i}} \left(I(\mathbf{x}_2 + \mathbf{i}) - \bar{I}(\mathbf{x}_2)\right)^2}}$$



Seed for propagation

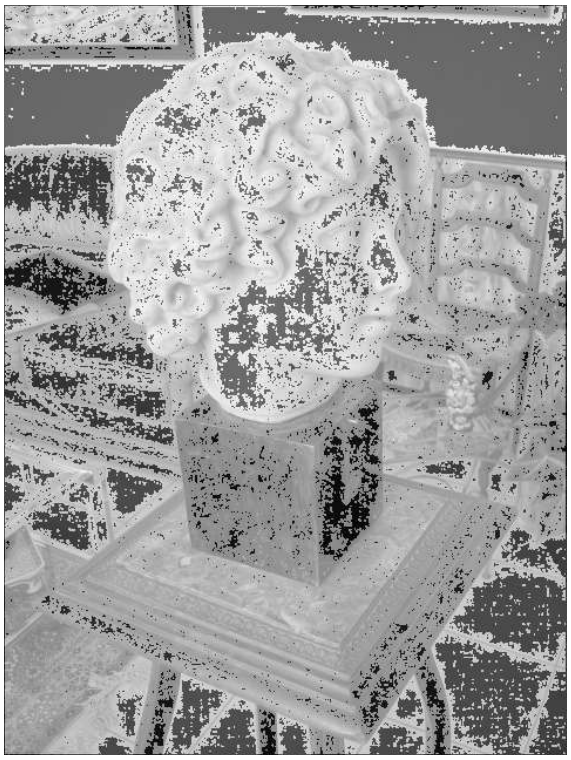


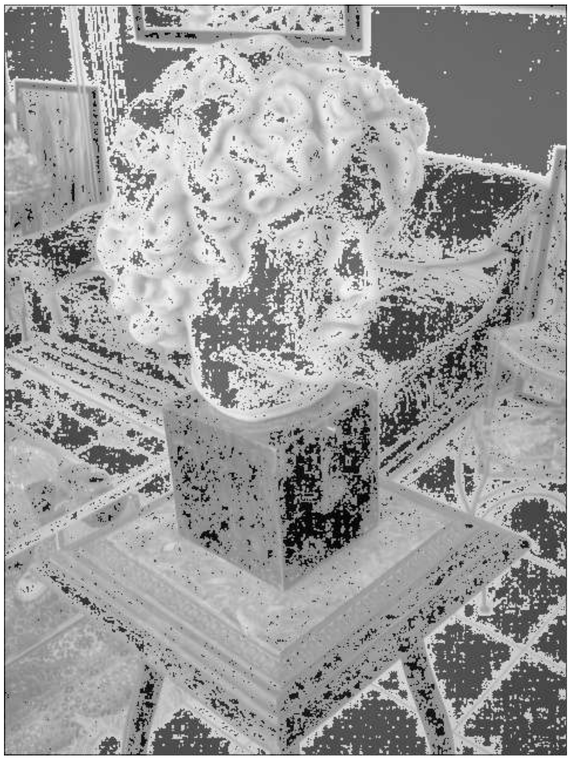
Matching Propagation (propagate.m)

- Maintain a priority queue Q
- Initialize: Put all seeds into Q with their ZNCC values as scores
- For each iteration:
  - Pop the match with best ZNCC score from Q
  - Add new potential matches in their immediate spatial neighborhood into Q
- Safety: handle uniqueness, and propagate only on matchable area

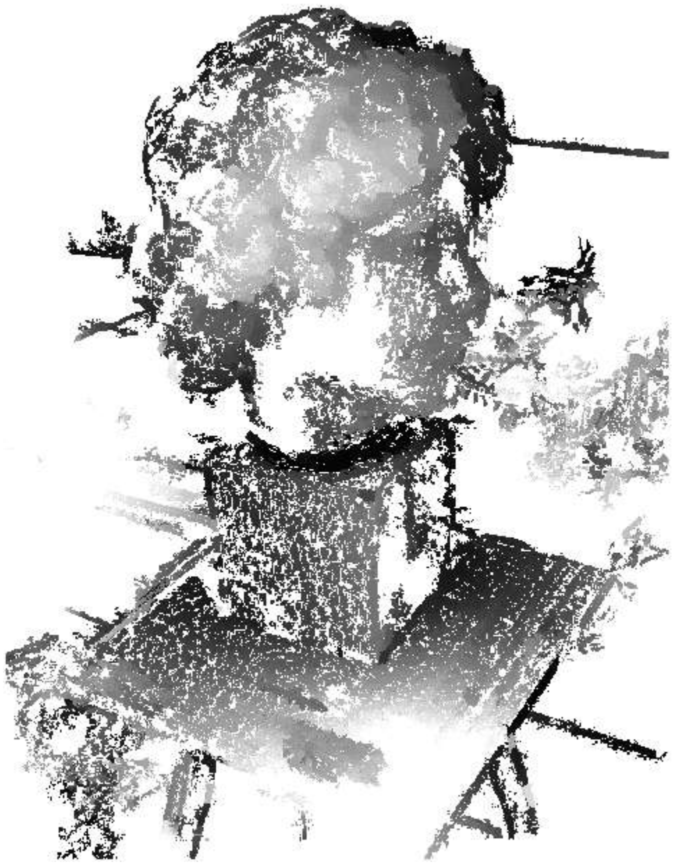
Matchable Area

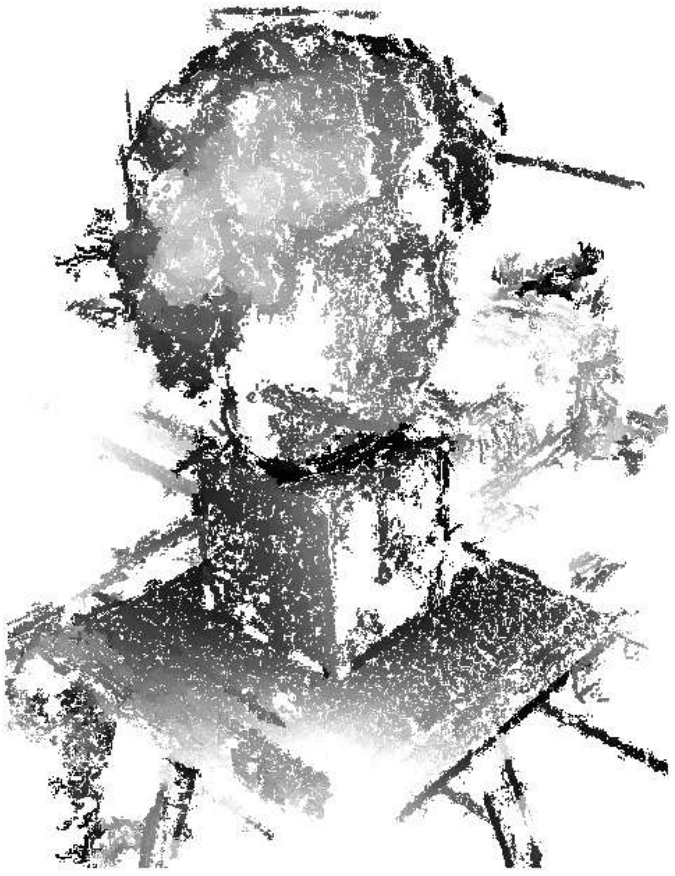
the area with maximal gradience > threshold



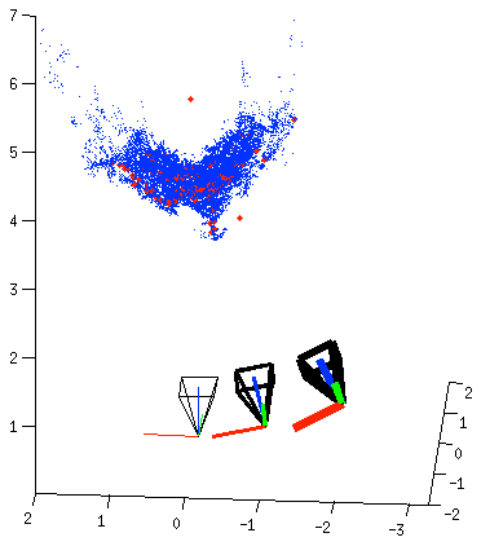


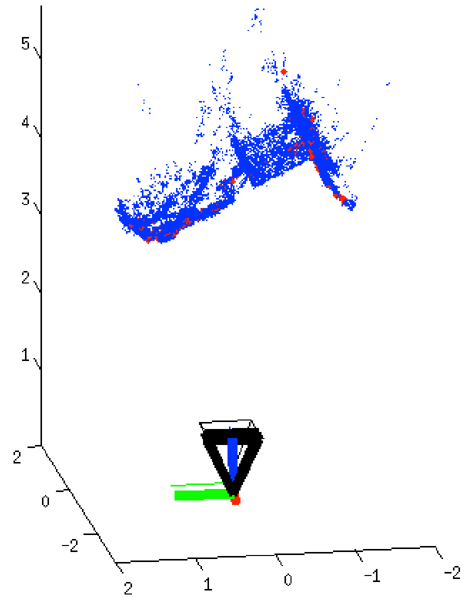
Result (denseMath/run.m)

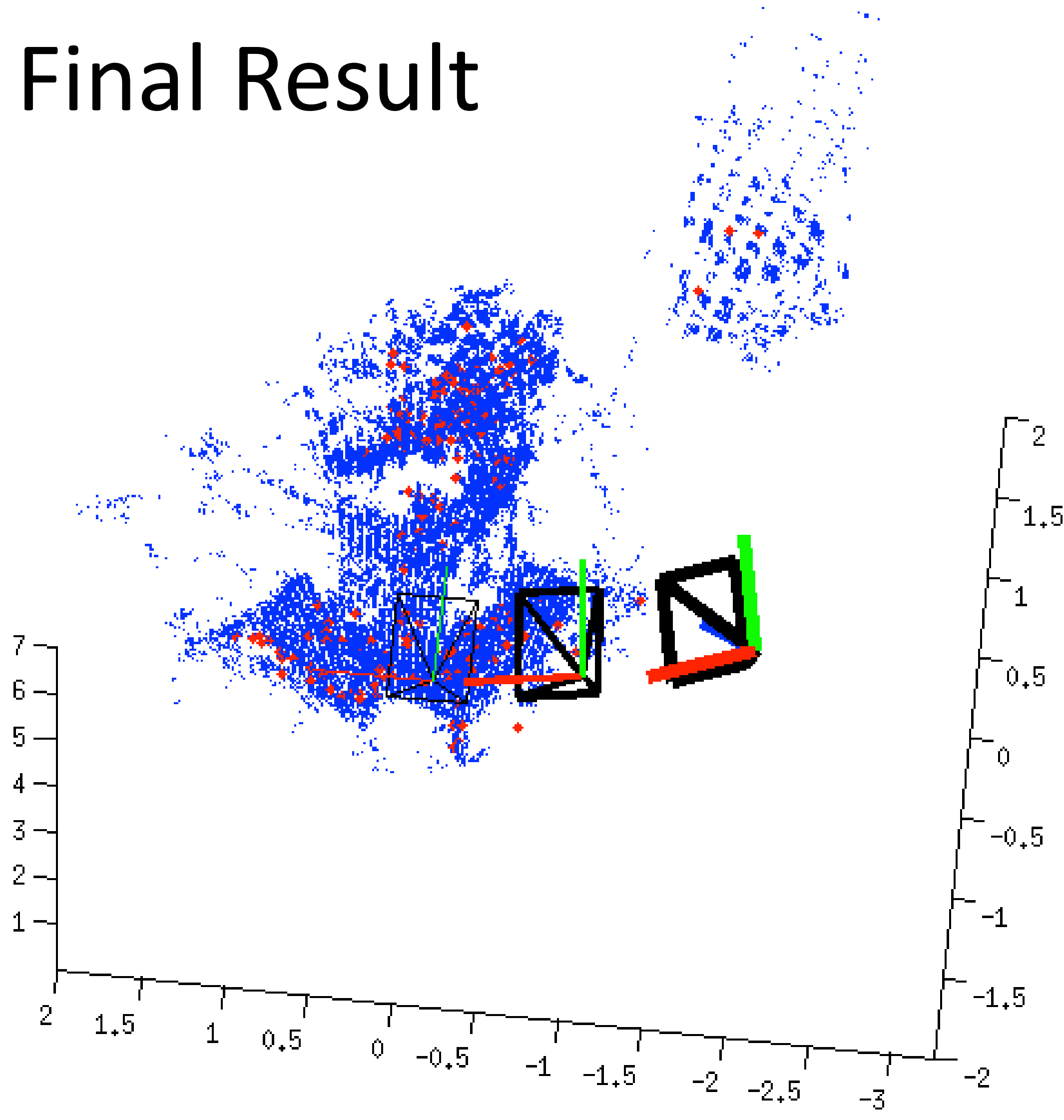




Final Result







Colorize the Point Cloud