A NEW MODELLING TECHNIQUE FOR OBJECT-ORIENTED PHOTOGRAMMETRIC COMPUTER VISION ALGORITHMS

H. Müller [a,*], E. Gülch [a], W. Mayr [a]

[a] Inpho GmbH, Smaragdweg 1, 70174 Stuttgart, Germany – (hardo.mueller, eberhard.guelch, werner.mayr)@inpho.de

Commission III

KEY WORDS: Mathematics, Design, Modelling, Algorithms, Development

ABSTRACT:

Photogrammetric Computer Vision algorithms often deal with complex numerical calculations and complex data structures. For modelling the numerical calculations usually a mathematical model is used. For the data structures an Object-Oriented model is appropriate. But there is still a lack in modelling the combination of both kinds of notation.
In this paper we present a new modelling technique for presenting mathematical and object-oriented aspects in one model. This modelling technique is based on the Unified Modelling Language (UML) and optimally suited for algorithms with complex numerical models and data models. For the semantics of class members in the UML model we use a formal description of constraints, which is similar to the Object Constraint Language (OCL) included in the UML specification. Since OCL is not capable of complex mathematical expressions, we enhance this notation to handle mathematics, and we name it OCL+M. We show, that typical problems in photogrammetry can be modelled by using an object-oriented model with OCL+M.
The advantages of this modelling technique are: 1. A clear presentation of the mathematical model within the object-oriented model, 2. The methods of the classes are easy to implement, 3. The implementation is straightforward and maintainable and 4. The model can be tested with little efforts.

1. INTRODUCTION

Photogrammetric Computer Vision algorithms often deal with complex numerical calculations at the one hand and with complex data structures at the other hand. For modelling the numerical calculations typically a mathematical model is used. For the data structures sometimes an Object-Oriented model is used. But there is still a lack in modelling the combination of both kinds of models. The relations between the mathematical model and the object-oriented model of a method for Computer Vision algorithms or similar topics are frequently unclear.

In this paper we present a new modelling technique for presenting mathematical and object-oriented aspects in one model. This modelling technique is UML based and optimally suited for algorithms with complex numerical and data models.

We show, that typical problems in Photogrammetry can be modelled using this technique and several examples are presented to illustrate it.

2. MATHEMATICAL AND OBJECT ORIENTED MODEL

To describe a complex system, a method or phenomena we need a model, which contains all relevant properties. A model is essential for communication between actors, who have to deal with it. Depending on the application field it should depict all relevant properties of structure and behaviour.

* Corresponding author.

2.1 The Mathematical Model

For most engineering and natural science applications a mathematical model is used. It consists usually of a set of algebraic expressions, which describe the coherences between the related features.

In the following example we have the mathematical model of a simple camera, which is described by the collinearity equations (Kraus, 1993).

$$k_x = c\frac{r_{11}(X-X_0)+r_{12}(Y-Y_0)+r_{13}(Z-Z_0)}{r_{31}(X-X_0)+r_{32}(Y-Y_0)+r_{33}(Z-Z_0)} \quad (1)$$

$$k_y = c\frac{r_{21}(X-X_0)+r_{22}(Y-Y_0)+r_{23}(Z-Z_0)}{r_{31}(X-X_0)+r_{32}(Y-Y_0)+r_{33}(Z-Z_0)} \quad (2)$$

where

X, Y, Z	= Coordinates of a point in object space
k_x, k_y	= Coordinates of a projected point
c	= Focal length
X_0, Y_0, Z_0	= Coordinates of the projection center
r_{ij}	= Elements of the rotation matrix

Using this model, we can see the relation between a point in object space and the pictured point in the focal plane of the camera. On the other hand it is not possible, to recognize only from the mathematical model, which elements are assignable as camera parameters (e.g. for camera calibration), if no additional semantic information is available.

In case of objects with a lot of attributes and a complex structure, e.g. the earth surface with a lot of diverse geographic features, it is not practicable to use only a mathematical model to express all properties. So, the mathematical model is limited and in some cases we need another model to describe the world.

2.2 The Object Oriented Model

Object Oriented Models are frequently used for software development, database, and GIS design as well as for business modelling. Real world objects are in this case represented as instances of classes with attributes and methods. The model elements can be depicted both using a graphical notation and as programming constructs in an object oriented programming language.

In the following examples we will use the Unified Modeling Language (UML) as a graphical notation, since it has become in the meantime a wide spread standard (Booch, 1999; OMG, 2001).

In case of the camera example the object oriented model of a simple camera is shown as an UML diagram in figure 1.

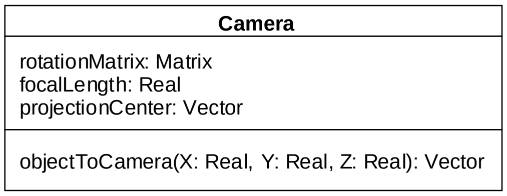

Figure 1. Object oriented model of a simple frame camera

From the model, it is clearly recognizable, which properties and functionalities belong to a camera, but it is not obvious, how these properties are used.

So, the object oriented model gives information, which is not in the mathematical model and vice versa. Thus, these two kinds of modeling techniques are not concurrent, but complement each other.

3. COMBINATION OF BOTH MODELS

To specify the semantics of the model elements in an object oriented model, the Object Constraint Language (OCL) has been included in the UML specification. With OCL expressions, one can model constraints for attributes or methods. It is also possible to express mathematical formulas in OCL, as shown in the case of the camera example:

```
context Camera::objectToCamera
           (X: Real, Y: Real, Z: Real): Vector
post:
let X0:Real  = self.projectionCenter.element(0)
let Y0:Real  = self.projectionCenter.element(1)
let Z0:Real  = self.projectionCenter.element(2)
let c:Real   = self.focalLength
let r11:Real = self.rotationMatrix.element(0,0)
let r12:Real = self.rotationMatrix.element(0,1)
let r13:Real = self.rotationMatrix.element(0,2)
let r21:Real = self.rotationMatrix.element(1,0)
let r22:Real = self.rotationMatrix.element(1,1)
let r23:Real = self.rotationMatrix.element(1,2)
let r31:Real = self.rotationMatrix.element(2,0)
let r32:Real = self.rotationMatrix.element(2,1)
let r33:Real = self.rotationMatrix.element(2,2)
result.element(0)=
         c * (r11*(X-X0)+r12*(Y-Y0)+r13*(Z-Z0))
           / (r31*(X-X0)+r32*(Y-Y0)+r33*(Z-Z0))
result.element(1)=
         c * (r21*(X-X0)+r22*(Y-Y0)+r23*(Z-Z0))
           / (r31*(X-X0)+r32*(Y-Y0)+r33*(Z-Z0))
```

The problem here is, that OCL was not designed for mathematical purposes, and so an OCL expression becomes quickly intricate as can be seen in the above example.

If the OCL would be extended with mathematical notation, one could express the camera example as follows:

```
context Camera::objectToCamera
              (X: Real, Y:Real, Z:Real): Vector
post:
let
```

$$X_0{:}Real = \text{self .projection Center .element(0)}$$

let

$$Y_0{:}Real = \text{self .projection Center .element(1)}$$

let

$$Z_0{:}Real = \text{self .projection Center .element(2)}$$

let

$$c{:}Real = self.focalLength$$

let

$$\mathbf{R : Matrix} = (r_{ij}) = \mathbf{self.rotationMatrix}$$

$$\mathbf{result} = c\begin{pmatrix} \dfrac{r_{11}(X - X_0) + r_{12}(Y - Y_0) + r_{13}(Z - Z_0)}{r_{31}(X - X_0) + r_{32}(Y - Y_0) + r_{33}(Z - Z_0)} \\ \\ \dfrac{r_{21}(X - X_0) + r_{22}(Y - Y_0) + r_{23}(Z - Z_0)}{r_{31}(X - X_0) + r_{32}(Y - Y_0) + r_{33}(Z - Z_0)} \end{pmatrix}$$

In this case the mathematical model is combined with the object-oriented model in an unambiguous way, and the form is well readable. Therefore, we suggest this approach in order to combine mathematical and object-oriented modeling to take advantage of both kinds of techniques. We name the extension of the OCL in the following ***OCL+M*** to distinguish it from pure OCL expressions.

4. EXAMPLE: AN EDGE EXTRACTOR

The usability of the combined mathematical and object-oriented model is shown by another example, in this case an edge extractor (Fuchs, 1998). The object-oriented model of the edge extractor is shown in figure 2. The model consists of a set of classes, whose methods are constrained to result in the mathematical relations of the edge extraction method. The most important elements are the `GrevalueImage` class, which represents a single channel image as input data, and the `EdgePixelImage` class with the `isEdgePixel` method, which represents among other things a binary image with edge pixels. This is articulated by OCL+M expressions in the following sections.

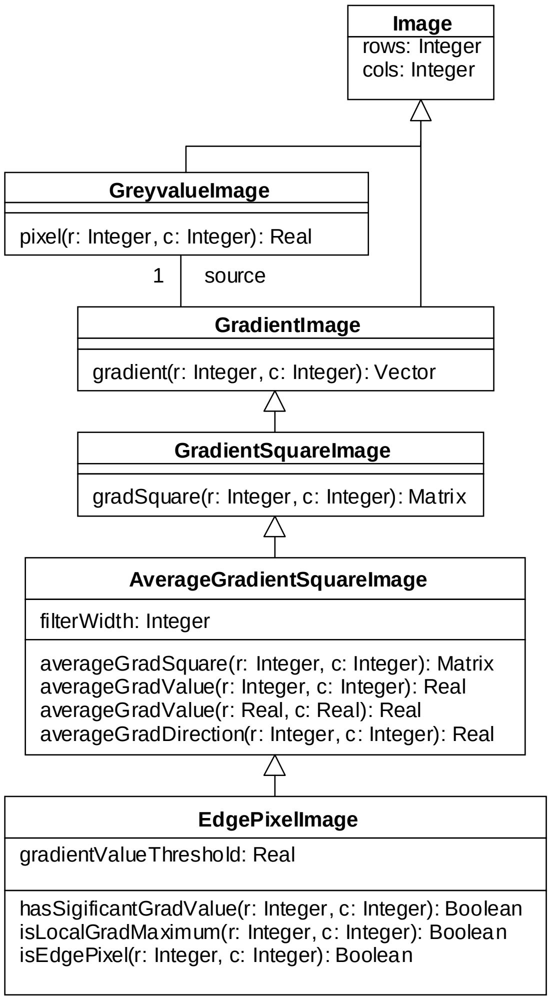

Image
rows: Integer
cols: Integer
GreyvalueImage
pixel(r: Integer, c: Integer): Real
1
source
GradientImage
gradient(r: Integer, c: Integer): Vector
GradientSquareImage
gradSquare(r: Integer, c: Integer): Matrix
AverageGradientSquareImage
filterWidth: Integer
averageGradSquare(r: Integer, c: Integer): Matrix
averageGradValue(r: Integer, c: Integer): Real
averageGradValue(r: Real, c: Real): Real
averageGradDirection(r: Integer, c: Integer): Real
EdgePixelImage
gradientValueThreshold: Real
hasSigificantGradValue(r: Integer, c: Integer): Boolean
isLocalGradMaximum(r: Integer, c: Integer): Boolean
isEdgePixel(r: Integer, c: Integer): Boolean

Figure 2. Object oriented model of an edge extractor

4.1 The Gradient Image

The first step of the edge extraction method is to calculate a gradient image. The `GradientImage` class uses for this purpose the Sobel operator and the result is an image with 2d-gradient Vectors as elements, as the following OCL+M expression shows:

```
context GradientImage::gradient
              (r: Integer, c: Integer): Vector
post:
let
```

$$\mathrm{g}(r\!: Integer, c\!: Integer)\!: Real = \mathrm{self.source.pixel}(r,c)$$

```
let
```

$$\mathbf{D}^{\mathbf{r}} = (d_{ij}^{r}) = \frac{1}{8}\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

```
let
```

$$\mathbf{D}^{\mathbf{c}} = (d_{ij}^{c}) = \frac{1}{8}\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

$$\mathbf{result} = \sum_{i=1}^{3}\sum_{j=1}^{3} g(r+i-2, c+j-2)\begin{pmatrix} d_{ij}^{r} \\ d_{ij}^{c} \end{pmatrix}$$

4.2 The Squared Gradient Image

From the `GradientImage` class, the `GradientSquareImage` class is derived, which extends it by a method for calculating the square of the gradients.

```
context GradientSquareImage::gradSquare
              (r: Integer, c: Integer): Matrix
let
```

$$\mathrm{g_r}(r\!:Integer, c\!:Integer)\!:Real = \mathbf{self.gradient(}r,c\mathbf{)e_1}$$

```
let
```

$$\mathrm{g_c}(r\!:Integer, c\!:Integer)\!:Real = \mathbf{self.gradient(}r,c\mathbf{)e_2}$$

```
post:
```

$$\mathbf{result} = \begin{pmatrix} \mathrm{g_r}(r,c)^2 & \mathrm{g_r}(r,c)\,\mathrm{g_c}(r,c) \\ \mathrm{g_r}(r,c)\,\mathrm{g_c}(r,c) & \mathrm{g_c}(r,c)^2 \end{pmatrix}$$

4.3 The Averaged and Squared Gradient Image

A box filter, whose width depends on the expected edge width, convolves the squared gradient image. This process is modelled in the method `averageGradSquare` of the `AverageGradientSquareImage` class.

```
context AverageGradientSquareImage def:
let
```

$$\mathbf{H(}r\!:Integer, c\!:Integer\mathbf{)\!: Matrix} = (h_{ij}) = \mathbf{self.averageGradSquare(}r,c\mathbf{)}$$

```
context
AverageGradientSquareImage::averageGradSquare
              (r: Integer, c: Integer): Matrix
post:
let
```

$$f = self.filterWidth$$

```
let
```

$$\mathbf{G(}r\!:Integer, c\!:Integer\mathbf{)\!: Matrix = self.gradSquare(}r,c\mathbf{)}$$

$$\mathbf{result} = \frac{1}{2f+1}\sum_{i=r-f}^{r+f}\sum_{j=c-f}^{c+f}\mathbf{G(r,c)}$$

The `averageGradValue` method returns the absolute value of the averaged gradient. This method is overloaded by a method with Real type parameters of the image position, which returns a bilinear interpolated value of the original method.

```
context
AverageGradientSquareImage::averageGradValue
               (r: Integer, c: Integer): Real
post:
```

$$result = \text{trace}\ \mathbf{H}(r,c)$$

```
context
AverageGradientSquareImage::averageGradValue
                      (r: Real, c: Real): Real
post:
let
```

$$\text{b}(r\text{: } Integer,\ c\text{: } Integer)\text{: } Real = \text{self .averageGradValue}(r,c)$$

```
let
```

$$r_0 : Integer = \text{floor}(r)$$

```
let
```

$$c_0 : Integer = \text{floor}(c)$$

$$result = \left(\begin{pmatrix} r_0 c_0 & r_0 & c_0 & 1 \\ r_0(c_0+1) & r_0 & c_0+1 & 1 \\ (r_0+1)c_0 & r_0+1 & c_0 & 1 \\ (r_0+1)(c_0+1) & r_0+1 & c_0+1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} b(r_0,c_0) \\ b(r_0,c_0+1) \\ b(r_0+1,c_0) \\ b(r_0+1,c_0+1) \end{pmatrix} \right)^T \begin{pmatrix} rc \\ r \\ c \\ 1 \end{pmatrix}$$

The last method of this class is `averageGradDirection`. It returns the direction of the average squared gradient.

```
context
AverageGradientSquareImage::averageGradDirection
               (r: Integer, c: Integer): Real
post:
```

$$result = \tfrac{1}{2}\arctan(2h_{21}(r,c), h_{11}(r,c) - h_{22}(r,c)) + \tfrac{\pi}{2}$$

4.4 The Edge Pixel Image

The `EdgePixelImage` class represents an image, whose pixels are Boolean values. They are true, if the pixel in the associated grey value image is an edge pixel.

```
context EdgePixelImage def:
let
```

$$\text{b}(r\text{: } Real,\ c\text{: } Real)\text{: } Real = \text{self .averageGradValue}(r,c)$$

One of the constraints of an edge pixel is that the absolute gradient has a significant value, which is tested by the method hasSignificantValue.

```
context EdgePixelImage::hasSigificantGradValue
               (r: Integer, c: Integer): Boolean
let
```

$$T_b = self.gradientValueThreshold$$

```
post:
```

$$result = \text{b}(r,c) > T_b$$

Another constraint of an edge pixel is that the absolute gradient value is a local maximum in the direction perpendicular to the edge. This is tested by the method `isLocalGradMaximum`.

```
context EdgePixelImage::isLocalGradMaximum
               (r: Integer, c: Integer): Boolean
post:
let
```

$$\phi(r : Integer, c : Integer) : \text{Real} = \text{self .averageGradDirection}(r,c)$$

$$result = \text{b}(r + \sin\phi(r,c), c + \cos\phi(r,c)) < \text{b}(r,c)$$
$$\text{and}\quad \text{b}(r - \sin\phi(r,c), c - \cos\phi(r,c)) < \text{b}(r,c)$$

The last method of this class is called isEdgeImage and returns whether the pixel is an edge pixel or not.

```
context EdgePixelImage::isEdgePixel
               (r: Integer, c: Integer): Boolean
post:
```

$$result = \text{self .hasSignificantGradValue}(r,c)$$
$$\text{and}\quad \text{self .isLocalGradMaximum}(r,c)$$

5. CONCLUSIONS

We have shown, that the combination of object-oriented and mathematical modelling leads to results, which are helpful for a lot of applications.

The advantages of this modelling technique are:

1. A clear presentation of the mathematical model within the object-oriented model. Using this technique it is self-evident, which mathematical expression belongs to which class, attribute or method in the object-oriented model and vice versa.
2. The methods of the classes are easy to implement, since their semantic is well defined. If an algorithm is modelled that way, the software development process is accelerated and results in high quality software.
3. The implementation is straightforward maintainable.
4. The model can be tested with little efforts, because the results of methods and values of attributes are exactly defined by the modelling technique.

Especially for software developers, it would be supportive, if a lot of algorithms and methods would be modelled in that way. But this technique may not only be used for software development purposes, since the utilization of the object-oriented model in combination with the mathematical model offers a clear and less unambiguous mapping of real world objects via the object-oriented model to the mathematical model.

REFERENCES

Booch, G., 1999. *The Unified Modelling Language User Guide.* Addison Wesley Longman, Inc.

Fuchs, C., 1998. *Extraktion polymorpher Bildstrukturen und ihre topologische und geometrische Gruppierung.* DGK- Reihe C 502, München

Kraus, K., 1993. *Photogrammetry*. Dümmler, Bonn, pp. 12-15

OMG, 2001. *OMG Unified Modelling Language Specification.* Version 1.4., September 2001, http://www.omg.org (accessed July 2002)