D3.1.3: Report on the methods underlying D3.1.1 (Software for detection of facade components) and D3.1.2 (grammatical descriptions of buildings from images)

K.U.Leuven - PSI – 1 January, 2010

IBBT vzw, Interdisciplinair Instituut voor Breedbandtechnologie
Gaston Crommenlaan 8 (bus 102), 9050 Gent, België • E info@ibbt.be • T +32 9 331 48 00 • F +32 9 331 48 05 • www.ibbt.be



Report WP 3.1.1: Software for detection of facade components

1 Gathering Training Data

For the detection of facade elements using supervised learning algorithms it is necessary to train a suitable detector on a representative data set containing samples of the classes to be detected. To gather this data set an annotation as been created to facilitate the annotation process (Figure 1).

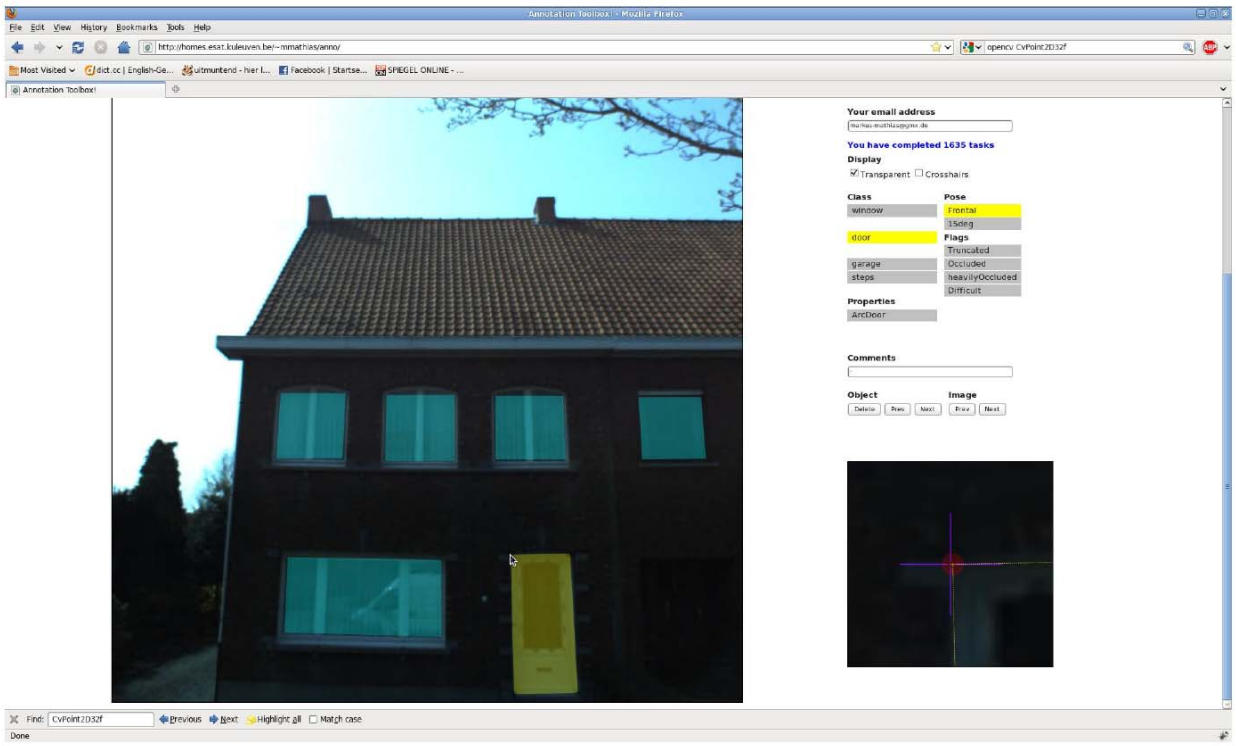


Figure 1: Web based annotation toolbox.

The annotation tool is based on the PASCAL VOC [1] annotation tool enriched with several features that are necessary to collect samples of the different classes with high quality annotations. Some of these features are:

- Annotation of rectangles and polygons for rectified and non-rectified images, respectively
- Rectification of images by warping based on the annotated polygons
- Zoom window to adjust the corners of the polygon with high accuracy

The performance of the detector is directly related to the annotation quality. But, the annotation of the different

object classes in the training images is not an easy task. There is a lot of variation of the elements to be annotated and annotators often have different interpretations of what to label in which way. To ensure consistent annotations by different annotators, we developed detailed annotation guidelines (see Appendix 1).

It must be noted that the tool is fully web browser based and supports the possibility to work together in groups to perform annotations.

Using the annotation tool more than 5500 objects of the categories of interest (windows, garages, doors, steps) have been marked in more than 2500 images. Every annotation contains additional flags about characteristics of the class (e.g. the window class is divided in “window with closed shutters”, “windows with arc-like structure on top”, “wide windows”, “ high windows”), about the pose (frontal or angled) and the appearance (truncated, occluded). Figure 2 shows the number of objects of the window class divided into its appearances and subcategories.

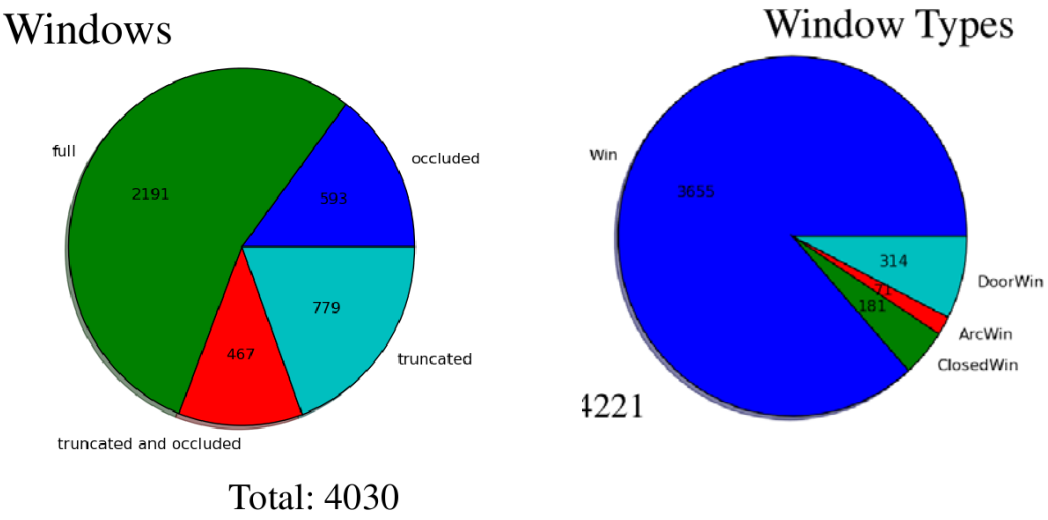


Figure 2: Annotation statistics.

2 The Object Detector:

Analyzing the problem of the detection of facade elements shows that many different classes of objects have to be detected (all main classes with their sub-categories) with a high intra-class variance and at the same time a high inter-class similarity. Figure 3 illustrates this problem for the window and door classes.

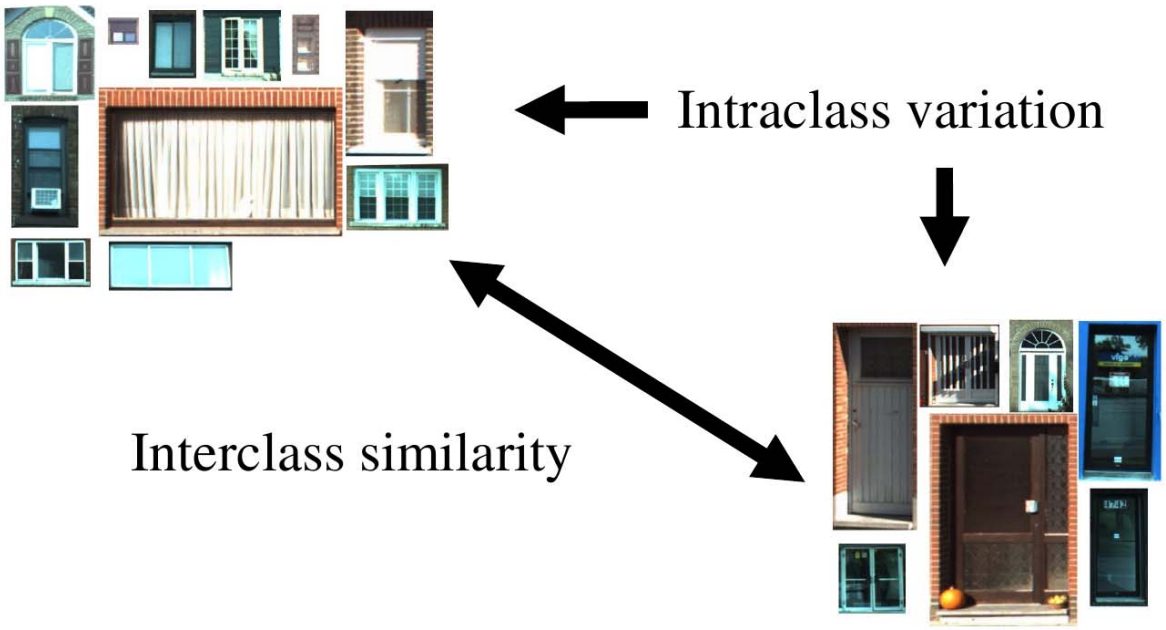


Figure 3: Variation of elements of the same class and similarity of element from different classes.

2.1 Adaboost Detector:

Looking only at the window class, a suitable detector should be able to handle (1) different categories of windows and (2) some similarity between these categories. The detection of a window itself is at this point more important than its categorization into the right sub-class. Schapire and Singer's Adaboost.MR [2] addresses exactly this multi-class multi-label problem, where multiple classes are handled but do not have to be mutually exclusive.

Algorithm 1 outlines the basic algorithm for the training of an Adaboost detector suitable for binary classification problems [3]. At each iteration the detector tries to find the feature that is most suitable in separating the classes of the training data. Depending on the quality of the feature (measured on the classification error) the selected feature gets more or less weight for the final detector decision. For the following iteration of the algorithm the training data is then re-weighted. Samples that have been miss-classified in the previous iteration get more weight in the following. In that way, the detector focuses more and more on the samples which are difficult to classify with the detector constructed up to the previous iteration.

For the multi-class Adaboost detector, the k-class problem is transferred into a two-class problem k times as large. Each input sample x from class l : (x, l) is derived into k samples:

((x, 1), 0), . . . , ((x, l), 1), . . . , ((x, k), 0) [2]

This detector still chooses the feature that separates the data the best for all classes together but selects an individual decision boundaries (thresholds) for each class against all other classes.

Given a set of training samples $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$

Search for a classifier: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Steps of the algorithm: for every iteration

- Find a feature that separates the data the best.
- Re-weight the training data: samples that are wrongly classified by using the selected feature get more weight, the others less. $\Rightarrow$ Concentration on "hard" samples.
- α_t is calculated based on the training error, it is a measure for the importance of the chosen Haar-like feature. The smaller the error the higher is α_t.

Algorithm 1: Basic Adaboost algorithm.

2.2 Adaboost Cascade

While considering all possible locations (bounding boxes) of an object of interest in an image, only very few of these locations will usually contain the object. With the initial version of the detector, the whole detector with all its features has to be evaluated for every possible bounding box to decide whether the given bounding box contains an instance of that object or not. A cascade of an Adaboost detector speeds up the detection process. Every location in the image is evaluated by a series of small classifiers. Each of theses small classifiers is tuned to sort out a lot (mostly 50%) of negative (background) samples and keep basically most of the positive samples (e.g. 99% of the windows). This series is called a cascade of classifiers (Figure 4). Most background samples are evaluated to be background at early stages in the cascade and only the real objects should go through all stages.

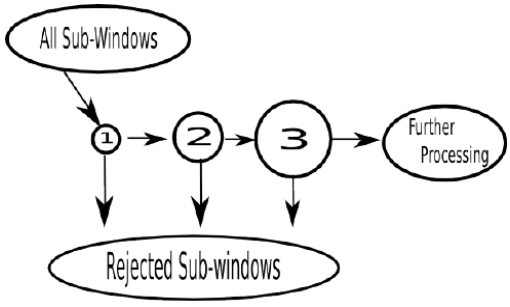


Figure 4: Schema of a Adaboost Cascade.

2.3 Sliding Window Detection with an Adaboost Cascade

The Adaboost Cascade can speed up detection tremendously (10 to 20 times faster). This gives reasonable detection speeds for example for faces or pedestrians. A bounding rectangle for a possible face/pedestrian has to be evaluated at all scales and positions. By speeding up the evaluation with the use of a cascade and integral images the detection can be done fast enough to be real time for these applications. In contrary to objects like windows the ratio of the evaluation window for pedestrians and faces varies very little. The evaluation of the detector at all positions, all scales and all possible ratios enlarges the search space that much, that an evaluation with this "sliding window" may take more than 2 minutes on a 640x480 (Intel core i7).

2.4 Improved Detection Using a Window Corner Detector

The detection task should be performed on previously rectified images, just like the images used for the training of the detector. This decision has been made because the majority of buildings are composed of rectangular structures that are aligned with the façade structure, e.g. windows, doors. Therefore, by considering only rectified façade images, the variance of facade objects appearances is reduced, and the chances of successful detection increased.

In the following, the focus will lie on the window detection task.

While the actual structure of the whole image does not always align to gradients in the image, the glass part of the window does. This fact can be used to reduce the search space for the detection. Figure 5 shows all the vertical and horizontal Hough lines detected in an input image. Once a line of a length larger than a threshold is detected, it is extended through the whole image. The intersection points of the vertical and horizontal lines are then taken as seeds for an intermediate detector, the Window Corner Detector.

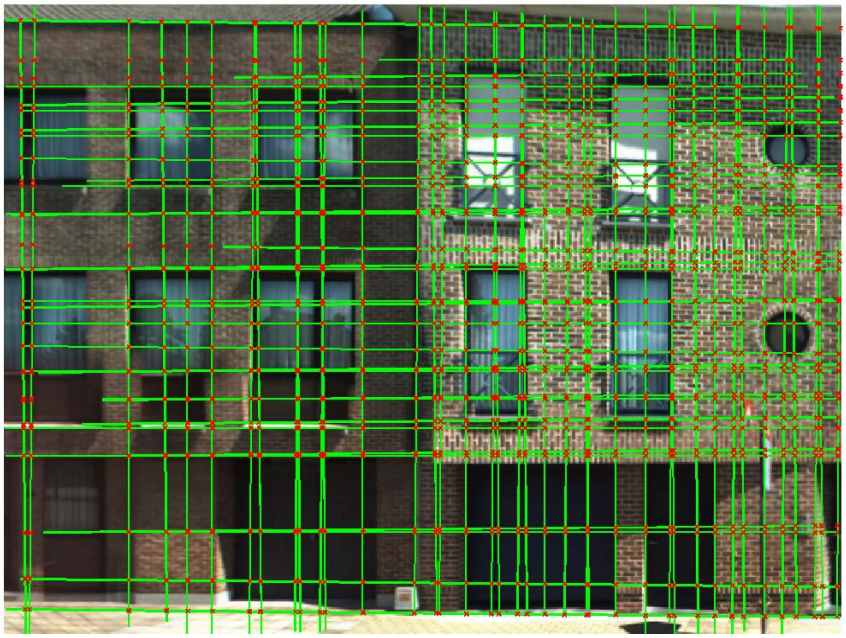


Figure 5: Facade with its horizontal and vertical Hough lines (green) and crossing points (red)

This Window Corner Detector consists of 4 individual cascaded Adaboost detectors trained separately on the four corners of the class of the window. The detectors are trained with rectangular regions around the window corners. The evaluation is only done at the seed point positions and at different scales. Points that pass this detection step will be marked as upper left, upper right, lower left and lower right corner points. Points can be detected as different window corners at the same time.

From this set of detected corner points, all possible rectangles that consist of at least 3 corners and fulfill some size requirements are added to the set of seed rectangles. Then, the actual window detector is evaluated for each of these seed rectangles.

It must be noted that most information required for successful window detection does not lie inside the glass part of the window but in its surroundings. Therefore, a detector trained on the glass will perform really poor. To circumvent that, the detector is trained for a bigger area than the glass part of the window. This area is found by analyzing the relative distances from the glass and the window structure bounding box in the training set.

To evaluate the detector on the seed rectangles, the same bounding box size offsets as those used for the training stage are used.

2.5 Re-Weighting of Detection Results

Even though most of the windows are detected, the false positive rate is very high if only the detector output is used (see Figure 6). By including additional information about facades the detector's output can be re-weighted with the goal of raising the value of the detector's response of correct windows while lowering the value of false detections. For this, the Adaboost detector's output can be used as a confidence value for the detection. After the re-weighting step non maximum suppression has to be performed to remove multiple detections. The re-weighting is split into global and local re-weighting.

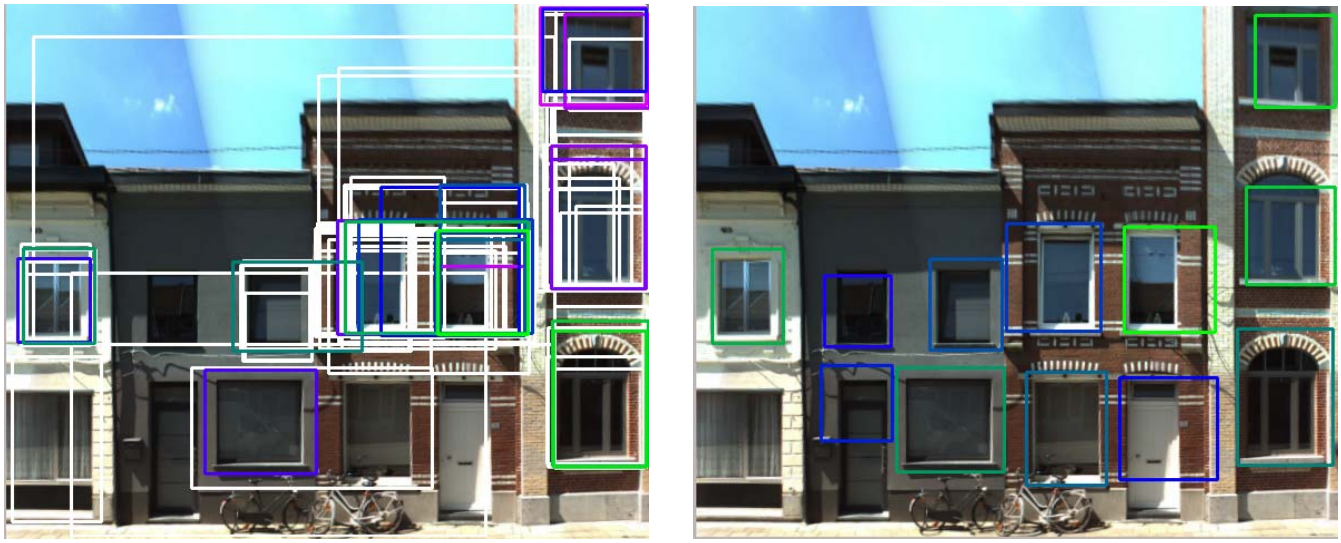


Figure 6: left: thresholded adaboost output, right: detected windows after reweighting.

2.5.1 Global Re-Weighting:

Global re-weightings considers the whole image or (if detected) the facade of a single house.

• Window alignment (horizontal and vertical): Windows of the same facade are usually aligned due to the organization of buildings into floors. This can be a strong prior for the positions where windows can appear in a facade. Windows with high confidence values that align horizontally or vertically can boost the other aligned windows confidence (Figure 7). The re-weighting is organized via a voting space, that considers all possible alignments and the confidences of all the windows participating on an alignment set. In that way, a window with low confidence can be rated higher because of many alignments with other windows in the same facade.

• Window position: if the façade boundary is known, windows that reach over the facades boundary are not allowed and the ones that touch the boundary should be less favorably re-weighted.

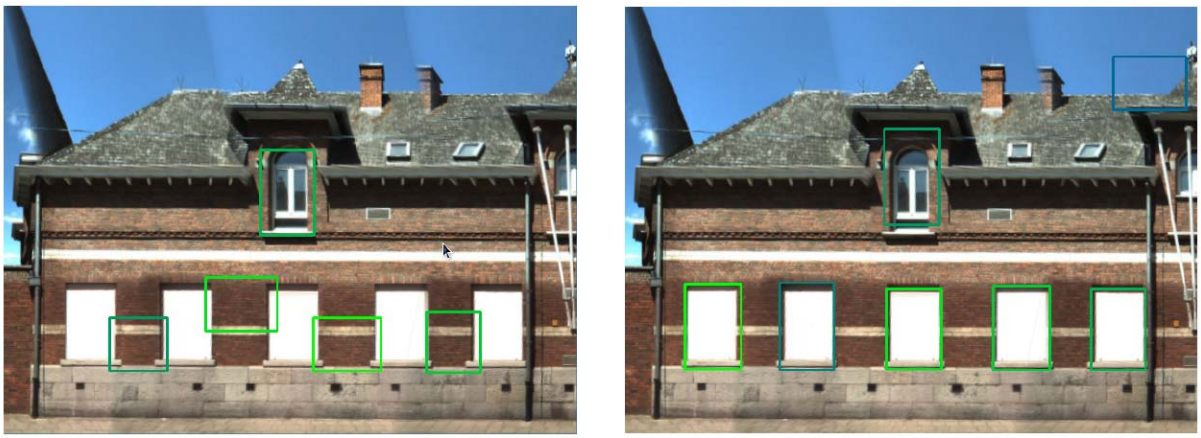


Figure 7: Global re-weighting gives more weight to window hypotheses that present some horizontal alignment.

2.5.2 Local re-weighting:

The detector's output is adjusted based on the properties of a single detection:

• Ratio (width/height): The ratio of the data in the training set is analyzed by Kernel density estimation. Figure 8 shows that ratios larger than 2.5 and smaller than 0.3 are very unlikely. The other ratios are rated accordingly

• Typical sizes of windows: As soon as the relation of pixels per meter is known, the confidence of very small or very big windows can be lowered. For example for windows higher than 2m or wider than 5 are unlikely.

• Vertical symmetry using Mutual Information (MI): windows often have a very strong vertical symmetry. Mutual Information can be used again to determine weather the left side of the window gives already a strong hint of the appearance of the right side

• Vertical line crossings: the confidence of detected windows is lowered when the top or the bottom of the window crosses a strong vertical is lowered (Figure 9).

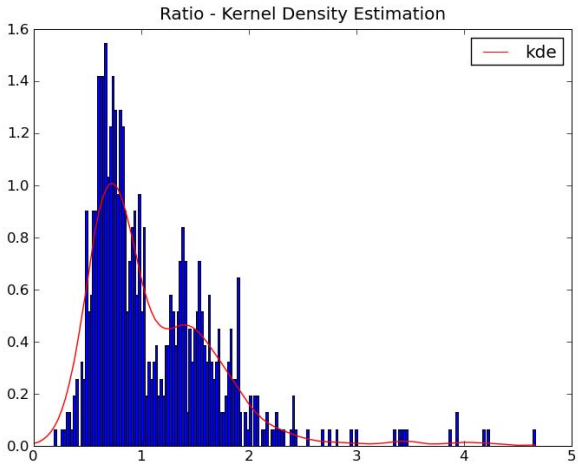


Figure 8: Kernel Density Estimation for window ratios.

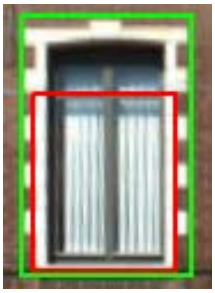


Figure 9: detections that do not cross vertical lines (green) are favored.

3 References:

[1] Everingham at al. “The 2009 PASCAL Visual Object Classes Challenge” (2009)

[2] Schapire, Singer “BoosTexter: A Boosting-based System for Text Categorization” (2000)

[3] Viola, Jones "Robust Real-time Object Detection” (2001)

Report WP 3.1.2:
Grammatical descriptions of buildings from images

Procedural modeling refers to the approach of creating models and texture from a set of rules. For this work, we express the rules using CGA shape, a split grammar that aims at iteratively refining the design by adding more and more detail [1]. We then use CityEngine [2] to visualize CGA-generated models. The derivation of the rules and the visualization by CityEngine are separated tasks.

In the following the derivation of procedural rules from an input image will be explained through an example.

First the components of the facade are detected using the detector described in WP 3.1.1. Then the building structure is derived form these detections. Finally this structure can be expressed by procedural rules and visualized using Procedural's CityEngine [2].

1 Extraction of building structure:

In order to use the detectors output the bounding boxes of the detected windows have to be aligned first. The alignment shown in Figure 10 helps to split the building into floors and tiles.



Figure 10: Window detections and their alignment

Procedural modeling is most powerful when repetitions exist. In the case of a building's façade, repetitions are expected with windows. Therefore, all detected windows should be compared with one another in order to identify similar ones that can then be described as instances of the same window type (with geometry and texture predefined in a window database).

The different windows of the facade are compared with each other using Mutual Information (MI). The mutual information between to images X and Y is defined as:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p_1(x)\, p_2(y)} \right)$$

It gives a measure of the information that both images share with each other, or, in other words, it measures how much the knowledge of one image reduces the uncertainty of the other one.

Similar windows are then replaced with the same window asset. This example window can be found bv matching, using MI, the similar windows with all window examples located in a database, as described in [2]. An example is given in Figure 11.

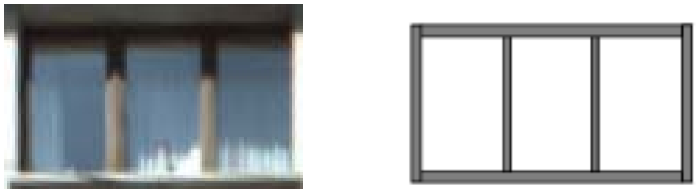


Figure 11: Window texture and asset selected from the asset database

Ground floors are typically more difficult to handle by procedural rules, because they can vary significantly from one building to the next. For example, suburban houses present ground floors that are very different than downtown buildings with ground-floor commercial activities. Furthermore, they are more likely occluded by cars, pedestrians, etc. As a result, we handle ground floors as single structures textured from the original image. As shown in Figure 12, the rest of the building is split into floors at the midpoint between the window rows. The areas between floor splits and window rows are called upper and lower ledge.

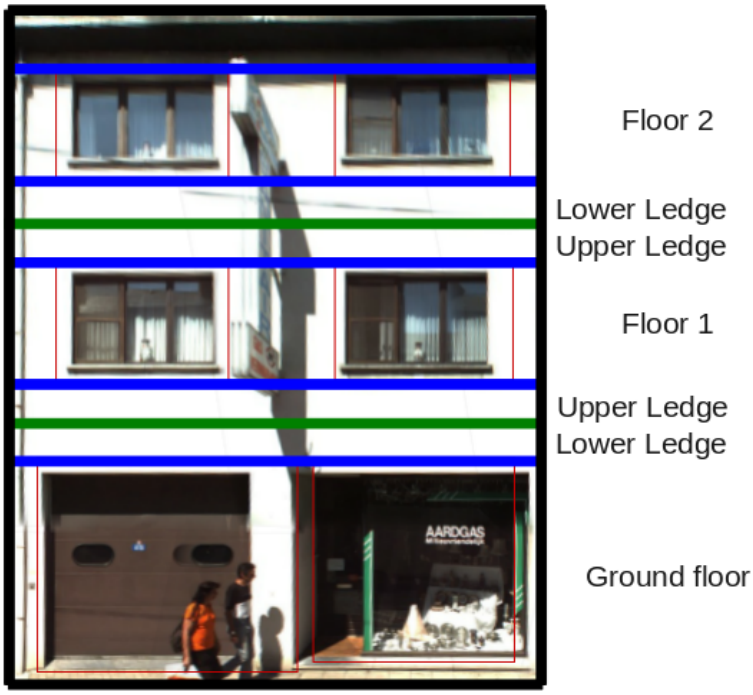


Figure 12: Horizontal split into floors and ledges

Both floors have a very high symmetry (MI) and can therefore be described together. Both are split into tiles (blue vertical line), then into windows and wall (Figure 13). The CGA shape grammar is highly based on splits and repletion and the subdivision of the façade into its parts can be expressed by its rules as described in the following.

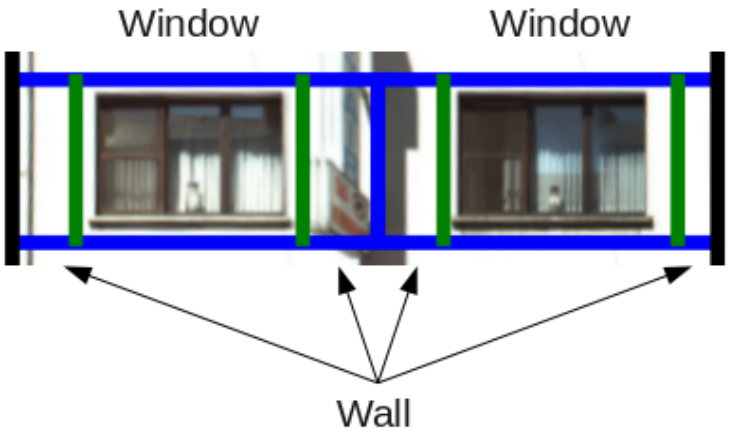


Figure 13: Vertical split of the floors into walls and windows

2 Formulation of procedural grammar rules:

The split grammar CGA shape has been inspired by both L-systems which have been used successfully for plant [4] and street network modeling [5] and shape grammars as introduced by Stiny[6] as a formal approach to architectural design. The derivation of Stiny's shape grammar, based on lines and points, has been simplified to set grammars, operating on arrangement of lines [6,3]. Furthermore the derivation process has been extended.

The shapes of the grammar are identified by symbols (strings), geometry and numeric attributes. Symbols that can be derived further are called non-terminal symbols. The production process ends if only terminal symbols are left. To modify the shapes, so called **scope rules** are defined by the grammar including: translation, rotation, scale etc. **Split rules** split scopes along one axis (for example facades into floors) or shapes into shapes of lower dimension (for example split the building bloc into its faces).

2.1.1 Excerpt of the derivation rules for the example building:

Figure 14 shows an excerpt of the derivation rules for a building (see also Figure 15). The *_height* and *_weight* variables refer to the corresponding facade element sizes measured in the image.

The derivation starts with the footprint of the building by extruding it to its height. "comp(f)" refers to the dimension reducing (component) split to select the front facade.

The front facade of the building is now split into its floors. Ground floor and the other floors are handled differently (ground floor has index 0). All the splits follow the description of the previous example image.

At this point, some symbols (*GroundFloorTexture, Wall*) can already directly be derived into their terminal symbols by referring for example to a texture (Figure 15, 3rd image refers to the given excerpt of the rules). Further rules describe the derivation of the *WindowTile*. The tile is composed by the original texture, a layer for adding some specularities and the previous shown window asset.

- *Init--> extrude(_height) Building*
- *Building --> comp(f) { front : Frontfacade}*
- *Frontfacade --> split(y){ groundfloor_height : Floor(0) |*
  ~floor_height : Floor(1) |
  ~floor_height : Floor(2) |
  0.5 : LedgeAsset }
- *Floor(floorindex) -->*
  case floorindex == 0 :
  split(y){ GroundFloorTexture | TopLedge}
  else :
  split(y){~wall_height: BottomLedge(floorindex) |
  ~window_height : Tile|
  ~wall_height : TopLedge }
- *Tile --> split(x){ ~1 : Wall | window_width : WindowTile | ~1 : Wall}*

Figure 14: Excerpt of the derivation rules for a building

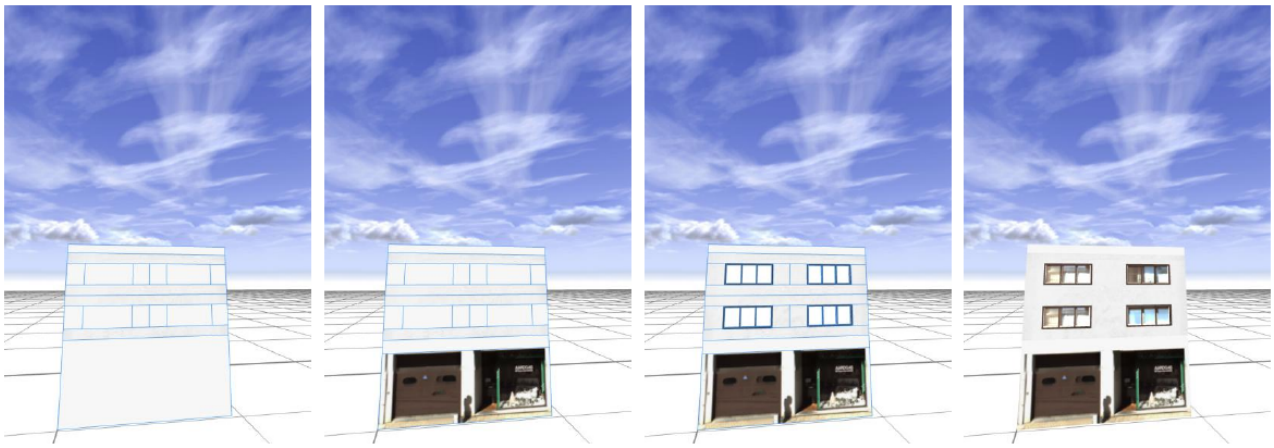


Figure 15: Rules are applied in a given order, the building takes form at increasing levels of detail.

2.2 Visualization with CityEngine:

CityEngine from Procedural is a powerful tool for 3D content creation and efficient visualization of 3D urban environments. It is used to visualize the CGA-described procedural model matched to the input image.

Further visualization improvements can be achieved by leveraging the semantic information contained in the procedural model. For instance, the model can be enriched by adding a reflective layer to the window and shifting the window slightly behind the facade plane (see for instance the right picture in Figure 16). These little refinements already enrich the resulting model of the given building without the usage of any further 3D information. Figure 16 shows an example of an input image and the resulting procedural model. Figure 17 shows the building from different angles showing the specularity effect visible in the windows of the model.



Figure 16: Input image and derived procedural model

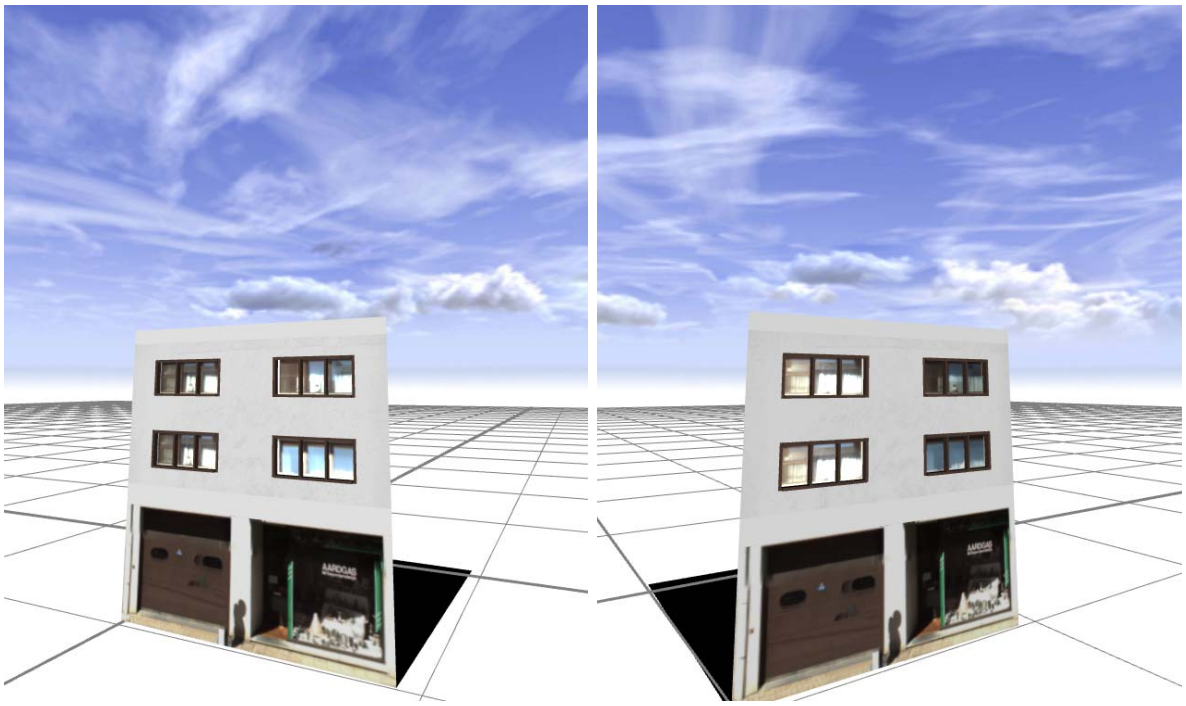


Figure 17: Procedural model from different angles

3 References:

[1] P. Müller, P. Wonka, P. Haegler, S. Ulmer and L. Van Gool " Procedural Modeling of Buildings"

[2] http://www.procedural.com/

[3] P. Müller, G. Zeng, P. Wonka and L. Van Gool "Image-based procedural modeling of facades" (2007)

[4] P. Prusinkiewicz and A. Lindenmayer "The Algorithmic Beauty of Plants" (1991)

[5] Y.I.H. Parish and P. Müller "Procedural modeling of cities" (2001)

[5] G. Stiny "Pictorial and Formal Aspects of Shape and Shape Grammars" (1975)

[6] P. Wonka, M. Wimmer, F. Sillion and W. Ribarsky "Instant architecture" (2003)

Appendix 1
Annotation Guide

URBAN Project Annotation Guide

Chris Engels Markus Mathias

October 12, 2009

1 Introduction

The goal of this annotation is to create a large dataset of structures commonly seen on building facades. The annotations consist of a bounding polygon and associated information, such as what the object is and how it is oriented. We will use this dataset to learn to automatically classify these structures using computer vision algorithms, with the end goal being to virtually reconstruct a semantically-labeled city in 3D. Because these algorithms can be sensitive to labeling errors and different interpretations of what should be annotated, we have developed this guide to ensure consistency throughout the dataset.

2 Facade structures

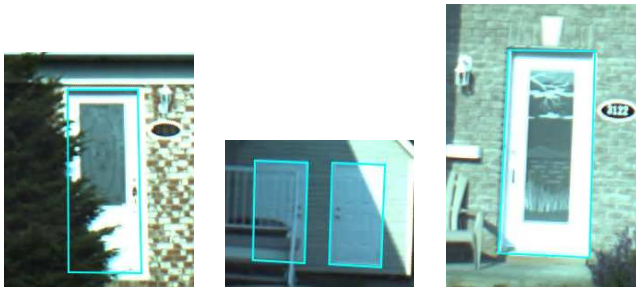
We are currently interested in annotations for the following structures: windows, doors, garages, and steps. Often these structures include frames or other supporting or aesthetic surroundings. We have adopted the following conventions to determine exactly what to annotate:

- **Windows:** Place the polygon corners on the corners of the glass when possible, and try to make the the edges of the polygon correspond as closely as possible to the edges of the glass. Only one polygon is necessary to cover all of the individual panes of glass, even if they can be opened independently, as long as they are set within a single larger frame. For irregularly shaped windows, follow the edges as much as possible while ensuring that the entire glass section is covered by the polygon. See Fig. 1 for examples.
- **Doors:** Place the polygon corners on the innermost frame of the door. A glass door or a door containing a window should still be labeled as a door. Windows inside the moving part of the door are not to be labeled. Windows that belong structually to the door should be labeled as window but with the property flag ‘door window‘. See Fig. 2 for examples
- **Garages:** Similarly to doors, place the polygon on the inner edge of the frame. For now, only annotate closed garage doors. See Fig. 3 for an example
- **Steps:** Only annotate the steps when there are at least two steps. Staircases should also be annotated. A rectangular bounding box is sufficient for steps.



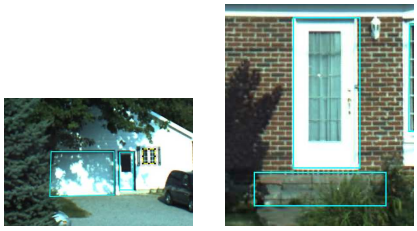
(a) (b) (c) (d)

Figure 1: Example window annotations. The corners of the bounding polygons should ideally be placed on the corners of the visible glass. (a) If the window is irregularly shaped, extend the polygon along the visible edges until the entire window is included. (b) Multiple windows within a single frame may be marked as one window. (c) When edges cannot indicate where a missing corner is, encapsulate as much of the window as possible and mark the object as truncated. (d) If edges are visible, completely mark the window.



(a) (b) (c)

Figure 2: Example door annotations.



(a) (b)

Figure 3: (a) An example of a garage. (b) An example of a step in front of a door.

3 Complicating factors

Unfortunately, many structures are partially or fully hidden by other objects such as trees, people, cars, etc. Given some bounding polygon, we define two separate cases to describe structures that are not fully visible: occlusion and truncation.

- **Occlusion** occurs when a significant portion of the structure within the bounding polygon is obscured by an intermediate object.
- **Truncation** occurs when a significant portion of the structure lies outside the bounding polygon, but is not visible.

Fig. 4 gives a few examples of these cases. An object should be considered occluded when at least 5% of the pixels lying within the polygon belong to an object other than the annotated structure. Heavily occluded means that more than 40% of the object is occluded. An object should be considered truncated when at least 15% of the object lies outside the polygon. Excessively truncated or occluded objects (under 30-40% of the object visible) may be left unlabeled. Occlusions are only physical objects blocking the view of the target, so shadows and overexposures should not be marked as occlusions. Another common complication is that many structures in the image will be too small to be useful. The annotation tool described in Sec. 6 will indicate the minimum size that is acceptable for a bounding box. Any structures below this size may be ignored. Finally, many facades will not face directly toward the camera. If the angle of the facade is more than 15° from the camera, it should be marked as non-frontal. Structures on facades that are more than 45° from the camera may be ignored.

4 Multi object annotation

Depending on the systems configuration you may work on multi object annotation or single object annotation. The annotation tool can be found at `http://homes.esat.kuleuven.be/~mmathias/anno`. To work with the tool login with the given username and password, enter your email address in the following page and press the button "Next". Make sure to use always the same address.

4.1 Mark the objects

First the objects in the image have to be marked according to the previously defined rules. There are two different ways to mark the objects:

- Draw a rectangular bounding box.
- Draw a Polygon.

Clicking and dragging draws a rectangular bounding box. As soon as the color of the box changes from red to green, the size is big enough and the box will stay visible after releasing the button. With single clicks a polygon will be drawn. To finish the polygon, use right click. Two clicks at close positions are counted only once, a polygon has to consist of at least three corners. The form of the



(a) (b) (c) (d)

Figure 4: Examples of difficult annotations. (a) The windows on the right should be labeled as non-frontal, while the top left window should be labeled as frontal but truncated. The window on the bottom left is too truncated to annotate. (b) This window is both truncated and heavily occluded. (c) The top window is occluded, the bottom window heavily occluded. (d) The structures on this facade are too angled to annotate and may be left unannotated.

objects can be changed by just clicking and dragging the resize handles. In case of a rectangle the resize handle in the middle of a line, move the whole line. As soon as the corner handles are moved, the rectangle changes to a polygon with only resize handles at the corners. To drop a corner, press "d" and then right click on the corner to drop. To delete the whole object, press "delete". Press the "shift" key to hide the annotated objects to get a better view of the image.

4.2 Class, Pose and Flags

After a object is marked give at least class and pose before continuing to annotated. Class, pose and flags should be selected based on the defined rules. The flags are accessible by the keyboard shortcuts "c", "v", "x", and "z".

4.3 Display

The check boxes "Transparent" and "Crosshairs" may help to better visualize the objects or the cursor position. They can be toggled on and of by clicking on the check boxes.

4.4 Navigation

The next and previous buttons in the object section allow to navigate trough the already annotated objects of the current image. The image section gives the possibility to get the "next" image (which might be either a new image or an already annotated image) or go back to earlier processed images. The shortcuts "space" and "n" select the next image and object respectively.

4.5 Tips

- In case of occlusion, follow visible edges of object, but do not extend polygon if the edge is not visible (find example)
- Self occlusions do not count as truncations or occlusions.
- Quality is more important than quantity.
- Don't be afraid to ask one of us if you are unsure of how to label an object.

5 Single object annotation

This tool works basically the same as the previous one, although it has some extensions.

5.1 Goal

The goal of this annotation step is to get a good rectification of the window and to annotate the window with all its sourroundings (not only the glass). If the rectification is good, it will be easy to draw a rectangular(!) boundingbox around the whole window structure (including shutter, windowsill, ornaments), otherwise the rectification has to be rectified.

5.2 Structure to annotate

- Windows: Annotate the whole window structure including shutters, windowsills, ornaments. Basically everything that belongs structurally to a window.
- Doors: Annotate the whole door structure including door-windows and first step but excluding all steps.
- Garages: Annotate the whole structure if present, otherwise draw a boundingbox that contains at least a bit of the sourrounding wall of the garage.

5.3 Workflow

1. Check the given label for the shown object. Usually the properties flags are not set yet. Press "showFullImage" if it is hard to see, what piece of facade you are looking at. If the object is misslabeled, press "changeLabel" to correct this.
2. Check if the whole object structure can be annotated correctly with a rectangular Boundingbox. If the horizontal and vertical lines of the structure are not parallel to the boundingbox, delete the bounding box again. If the boundingbox is drawn correctly, verify the automatically setted labels and go to the next image. Be aware that, the flags for occlusion and truncation might differ between the both bounding boxes of the object.

3. Adjust the annotation of the glass/door/garage and press "rectify". This should warp the annotated polygone to a rectangular one. Sometimes it helps to find a bigger structure with horizontal and vertical lines that can be used to rectify the image correctly. Go back to 2.