Forschungsseminar aus Computergraphik und Digitaler Bildverarbeitung (WS2009)
Michael Wimmer (Editor)

Symmetry detection in architectural meshes

C. Niederreiter

Institut für Computergrafik und digitale Bildverarbeitung, TU Wien, Austria

Abstract

3D city simulations or city planning applications often require a huge number of distinct architectural models in order to appear realistic. One way to generate many different models is to provide some template files to a computer program which automatically performs changes on the template in that it exchanges window elements, doors or structure elements of the facade and stores the result in a model database. But many models are available in the form of monolithic meshes without any information about their structure or exchangable parts. I pursued which methods are applicable for detection of structure elements which occur at least two times and implemented two approaches, a flood-fill based approach and a clustering based approach. The approaches regard local geometric characteristics in order to calculate vertex signatures that describe surface points of the mesh in a comparable way. Signature comparison yields evidences for symmetry relations that may be part of a larger symmetry relation between two instances of a structure element such as a window or door.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Hierarchy and geometric transformations

1. Introduction

Geometric meshes often contain information that is not directly accessible. For instance, meshes constructed by computer graphics modelers using modeling tools such as *3D Studio Max* mostly do not include any usable semantic data. Also, 3D scanning devices that digitize real-world surfaces are not able to produce semantic information. But semantic information is important to make geometric data alterable. Humans are inherently able to extract it from $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ projections even if it is not a known part of the geometric data file. They are swift in cognizing structural regularities such as symmetry relations, which are a form of semantic information.

This paper is aimed at finding ways to extract such information from architectural meshes in a computer process. Primarily, this is useful for computer graphics applications that require a huge number of virtual buildings which are based on few manually drawn building models that are used as *template objects* and are to be altered in order to gain an appreciable variety of buildings. This is often the case if large conglomerates of buildings such as cities are to be rendered, based on real cities where each house looks slightly different from each other. A suchlike city structure is used in [WMWG09], but this paper does not go into details concerning the modelling of buildings.

A possible solution for creating city structures is to follow a *hierarchical approach* from the very first (i.e. *bottom-up*): [WWSR03] introduce the term *split grammar*, characterized as a *set grammar* that is able to balance automated construction and fulfillment of user-defined presettings. Alike, [Mer07] describes a further way to achieve automated object synthesis. Both approaches operate on 2D or 3D shape elements and put them together according to apt rules.

On the contrary, monolithic meshes require a *top-down* approach regarding the entire mesh. The pertinent objective is to find segmentations of the mesh the cells of which consist of sensibly exchangeable parts of the mesh. That is, for architectural models, the cells represent windows, doors and possibly other structure elements. According cells are surface areas on the mesh that may be findable only if they can be uniquely described. Since the detection process should operate fully automated on arbitrary architectural models, it is restricted to detection of *geometric regularities*, but not user-defined feature characterisics. The term "geometric regularities" can be split into the subclasses *symmetry relations* and *similarity transforms*. Symmetry relations can oc-

cur within a cell (mirror symmetry in double-shuttered windows), but also between cells (similar-looking windows or doors).

The following section informs about the objectives that are to be achieved. Afterwards, two approaches for implementations are proposed, the first of which developed during my work on the topic.

The second approach this paper introduces tries to extend [MGP06] which describes a way of finding symmetries in samples taken from a geometric object with a *smooth surface*, such as an animal. This is achieved by regarding local shape descriptors of surface points that depend on unique main curvature directions in order to compute point-specific characteristics called *signatures*. However, most architectural objects primarily consist of orthogonal or other non-smooth elements that contain surfaces the derivation of which is non-continuous, or vertices in plain surfaces, so either the curvature calculation cannot be adopted as is or, worse yet, main curvatures cannot even be calculated because they are both 0 (in some respects such surface points are umbilical).

Finally, the extensible software framework used to investigate the problem and test the approaches is introduced.

2. Objectives

A typical architectural object is stored in a geometric object file according to the *3DS*, *OBJ* or *BLEND* file format specifications. There are more handy and program-independent alternatives to these formats publicly specified and available on the Web that are *human-readable*, namely *COLLADA* and *Ogre MESH XML*, which basically provide capabilities similar or even superior to the capabilities of aforementioned formats, but, anyway, are better suited for analytic purposes when speed is not of major importance. Sometimes, they are not recommended for end-user applications because of their performance drawbacks (decompression of packed files and parsing) or high storage overhead.

A COLLADA file fulfills the *XML* specification and stores the vertices $\mathbf{x}_i \in \mathbb{R}^3$ of one or several geometric meshes, as well as their normal vectors $\mathbf{n}_j \in \mathbb{R}^3$ that specify the surface normal directly at the vertex. The vertices of a mesh and also the normal vectors live in distinct (independent) and unstructured ASCII-arrays and are indexed by polygon structures, usually representing triangles, consisting of integer scalars, the indices pointing to the vectors in the vertex-array that describe the polygon's corner locations. Furthermore, references to texture images and texture mapping coordinates $(u,v) \in \mathbb{R}^2$ are usually part of the file.

As the polygon corners are indexed, even a COLLADA file is a trade-off between human-readability and computational performance, but they are far from a usable data structure that allows *identification of semantically sensible parts* of the mesh. So the first objective is to transform the file contents in the form of their structure to an analyzable form in the computer's memory. This task is to be performed by the *model loader*. If the file contains several meshes, these might be parts of an architectural object that is already split up in a semantic manner. This should be regarded when the model is loaded: The model loader should be cautious if it intends to join the submeshes, otherwise, semantic information gets lost.

In order to gain as much information about symmetry relations in a mesh as possible, it is necessary to discover its local features. Cross-polygon neighborhood relations of vertices can be used to calculate vertex *signatures* that mathematically describe the vertices, as well as the local *surface curvature* can contribute to the signature. Vertices differ in their neighbor vertex counts, neighbor distances and relative neighbor locations. If two vertices or surface points on a mesh have similar signatures, they might be part of a *symmetry relation*: A symmetry relation consists of more than just relations between individual points, rather it consists of *many point relations between at least two groups of points*. These point groups are linked by groups of similar *transformations*, i.e. if a transformation that links two groups is applied to an arbitrary point that is part of the first group, the point moves close to one of the points with a similar signature that should live in the other group.

One objective of the second, *curvature-based* approach introduced in this paper is to find transformations that are part of a symmetry relation and finally drop transformations that are not part of large-scale symmetry relations.

The most general transformation (*homography* or *projective transformation*) can be described using an invertible *4x4 matrix*. But since it is necessary to compare transformations in order to group them, a more compressed representation is desirable. According to [MGP06], a vector $\mathbf{x} \in \mathbb{R}^7$ is sufficient to describe a transformation that contains translation ($\mathbb{R}^3$), rotation ($\mathbb{R}^3$) and a scale factor ($\mathbb{R}$).

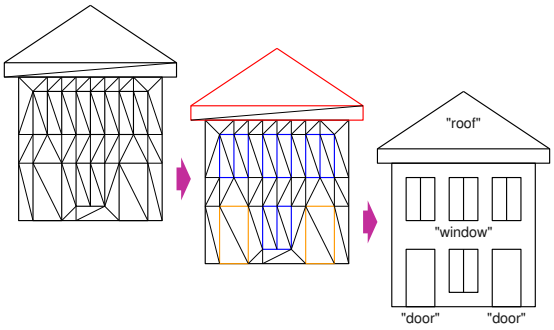

Figure 1: *From mesh to semantic information*

The grouping of transformations is a clustering task. Since

the number of groups is unknown (number of structure elements that are to be found, e.g. windows), the clustering algorithm by itself must be capable of detecting the right number of clusters in order to avoid user interaction.

The final objective is the appropriate naming (and classification) of detected structure elements, as drafted in figure 1. Naming means attachment of *verbal semantic information*. This is to be done via human interaction, since a computer program is not always capable of robustly guessing the meaning of individual structure elements, whereas humans are quick and reliable in doing so. [AMAK07] examine the problem of semantic labelling of 2D image parts. Labels such as "sky" or "person" are attached to image regions after matching a region's calculated *visual descriptor* with visual descriptors from a database. This approach could be adopted for labelling structure elements of buildings. In fact, it is not possible to define a visual descriptor, for instance for a *door*, because in reality, the visual appearance of doors overlaps with the visual appearance of windows. But considering "typical" doors and "typical" windows having different visual descriptors might be sensible. Bounding-box aspect ratios or texture image based visual descriptors are conceivable for automatic pre-classification. Subsequent precise labelling is to be done by the user.

3. Approaches

3.1. Simple Approach

Basically, this approach features a flood-fill procedure applied on vertices that is controlled by vertex signature comparison. The flood-fill's result is a group of vertices and further groups of vertices that form a structure element on the architectural mesh. Detection of a structure element usually relies on its multiple existence on the model, i.e. there must be at least two instances at different locations.

The following steps are to be performed in order to find structure elements on a mesh:

1. Model loading
   a. Find edges as well as accordant neighboring vertices for every vertex of the mesh or submeshes. Every vertex is stored in an object that contains the vertex coordinates as well as references to the edges and neighboring vertices.
   b. *Compile vertex signatures.* Two types of signatures are calculated: A *short signature* $s_s \in \mathbb{R}$ and a *long signature* which consists of n vectors $\mathbf{x}_i \in \mathbb{R}^3$, where n is the number of direct neighboring vertices (the path to the neighboring vertex is exactly one edge). The former contains the sum of *Manhattan distances* to the neighboring vertices $\sum_{i=1}^{n} \delta_x + \delta_y + \delta_z$ and can be used for quick preselection of presumably similar long signatures. The latter describes the direct neighboring relations of a vertex in detail. It contains the neighbor count n and the distance vectors $\mathbf{d}_i \in \mathbb{R}^3$ to the neighbors.
   c. *Signature comparison and accumulation.* If the mesh does not contain more than 50000 vertices (this value is estimated based on current hardware capabilities), it is recommendable to accumulate vertices that have similar signatures for every vertex of the mesh as final step of the model loading process. Signature comparison is explained in detail in the section *3.1.1 Signature comparison and accumulation*.
2. Structure detection
   a. *Flood-fill.* Starting with a single vertex $\mathbf{v}_i$ on a mesh or submesh, all neighboring vertices are visited. For vertex $\mathbf{v}_i$ and for each neighboring vertex $\mathbf{v}_j$, if not already performed in *step 1.c.*, vertices with similar signatures are searched on the entire mesh. The number $n_i + 1$ of similar-signatured vertices inclusive of the vertex $\mathbf{v}_i$, or an integer multiple or fraction of $n_i + 1$, if greater than a threshold $t \geq 1$ ($t \in \mathbb{N}$), indicates whether a vertex $\mathbf{v}_j$ with $n_j + 1$ similar-signatured vertices (inclusive of $\mathbf{v}_j$) is part of the same structure element or not. If it is, i.e. $x_{similar} = true$ (equation 2), the vertex is used as a new flood-fill seed point in a recursive procedure. The threshold t requires human interaction and can be preset to 1.

$$m_{min} = \min(n_i, n_j) + 1 \quad m_{max} = \max(n_i, n_j) + 1 \quad (1)$$

$$x_{similar} = \begin{cases} true & \text{if } m_{min} > t \wedge 0 \equiv m_{max} \bmod m_{min}, \\ false & \text{else.} \end{cases} \quad (2)$$

   In order to make this step more robust, the number of *neighbors* with the same number of similar-signatured vertices can be compared to a minimum threshold parameter $t_{simneigh,min}$, so, for instance, only if a neighboring vertex has at least $t_{simneigh,min}$ neighbors with the same number of similar signatures, it is considered a part of the same structure.
   Submeshes are usually not connected to the main submesh or the other submeshes and often smaller than a sensibly exchangable structure element, i.e. they are part of a structure element together with further submeshes, for instance window sills. To overcome this, a *sparkover distance* (vertex-to-plane or plane-to-plane) could be defined interactively, that allows the flood-fill process to change the mesh. A spatial data structure such as an *octree* can restrict the sparkover search to nearby submeshes and thus enhance performance.
   b. *Structure element selection.* In order to make individual structure elements selectable or exchangable as a whole, the flood fill must be performed on each instance of the structure element.

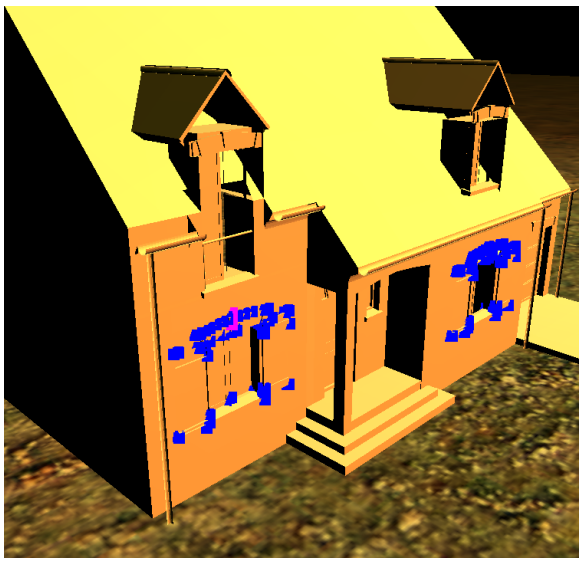

Figure 2: *Simple approach: The pink vertex is the user-selected seed point*

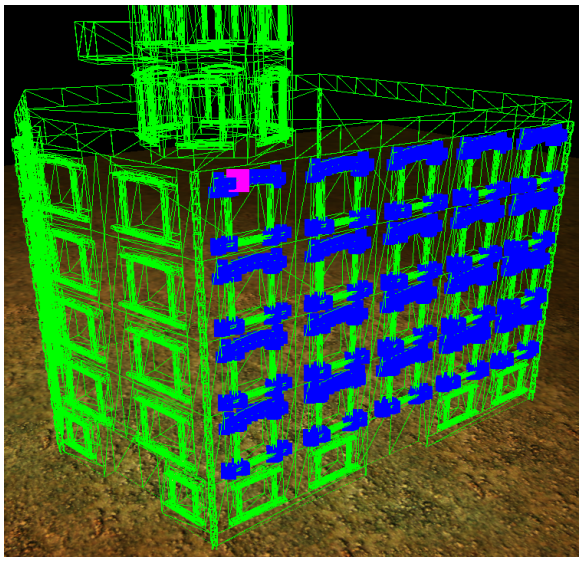

Figure 3: *Simple approach: Frequent occurrence of a structure element (window)*

3.1.1. Signature comparison and accumulation

Short signatures as described above in *step 1.b.* are used to compare and select vertices in a quick manner. Vertices that have similar short signatures (within a predefined range of tolerance) are evidence for a symmetry relation. This evidence is confirmed or revoked by comparing the long signature values (also within a predefined range of tolerance).

3.1.2. Results

Figures 2 and 3 show the results of this approach. The pink square is the user-selected seed point and the blue squares mark the vertices representing the structure elements.

3.1.3. Drawbacks of this approach

Robustness. Sometimes, the number of similar signatures is not sufficient for sensible structure detection and the flood-fill expands too far. Thus, a more robust type of signature is needed or even a more sophisticated approach than the flood-fill method. A superior solution is described in the next section.

3.2. Curvature-based approach

This approach is an adoption of the technique introduced in [MGP06], which works best with smooth surfaces.

3.2.1. Surface descriptor

Unlike the signatures used in the previous approach, signatures exclusively contain information about the *second derivative of the surface* of the mesh at a point on the surface, more precisely, they consist of the *principal curvatures*. Smooth meshes as well as non-smooth meshes are merely rough approximations of imaginary smooth surfaces, so the principal curvatures need to be estimated. [MGP06] suggest the method described in [ACSD*03] which simultaneously estimates principal curvatures and their orientation vectors as well as the surface normal at a given vertex based on the surface normals at surrounding surface points.

3.2.2. Procedure

Vertices are chosen as surface sample points. For each point, a surface descriptor as described above is calculated. The surface descriptor consists of the principal curvatures κ_1 (smaller curvature) and κ_2 (greater curvature). Umbilic points with $\kappa_1 = \kappa_2$ provide less comparable information, therefore points that feature a large κ_1/κ_2 ratio are omitted.

The next step is called *pairing*. Two vertices i and j are regarded at a time. Vertices that passed the previous selection based on the curvature ratio are iteratively connected to *all other vertices* that passed the selection. Vertices that provide evidence for a symmetry relation must have similar principal curvatures, other vertex pairs are rejected. Both $\kappa_{i,1}$ and $\kappa_{i,2}$ are compared to $\kappa_{j,1}$ and $\kappa_{j,2}$, respectively. Vertices that pass this test are connected by a seven-dimensional vector containing a scale component $s \in \mathbb{R}$, a rotational component $\mathbf{R} \in \mathbb{R}^3$ and a translational component $\mathbf{t} \in \mathbb{R}^3$. The scale component is estimated by comparing both principal curvatures between the vertices i and j: $s = \frac{\kappa_{i,1}/\kappa_{j,1}+\kappa_{i,2}/\kappa_{j,2}}{2}$. The

rotational component provides three euler angles that can be applied to rotate the surface patch around the vertex i so that the main curvatures of the patch are oriented similar to the main curvatures of the patch around vertex j. The translational component describes the move from vertex i to vertex j.

After the pairing step, each regarded vertex is connected to each other vertex by a transformation $\mathbf{T} \in \mathbb{R}^7$. Vertex transformations that go from one instance of a structure element to another instance of the same structure element form a cluster in $\mathbb{R}^7$. Thus, in order to identify symmetry relations, it is necessary to find clusters of transformations. The *k-means clustering algorithm* is very popular but cannot be used for this purpose, because the number of clusters is not known a priori and also cannot be guessed easily by the user. [MGP06] recommend the *Mean-shift* clustering algorithm, a non-parametric clustering algorithm which is treated in detail and enhanced by [CMM02] for use in image processing applications. The mean-shift algorithm is explained in the next section.

For further details about the general procedure please refer to [MGP06].

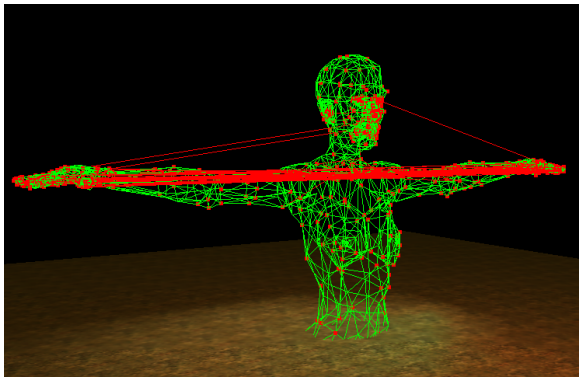

Figure 4: *Transformations between pairs of points after clustering on a smooth mesh*

3.2.3. Mean-shift clustering

Mean-shift clustering is a gradient-based clustering method. Unlike the k-means clustering algorithm that requires the number of seed points, i.e. the number of clusters to be known, the mean-shift algorithm relies on automatic seed point selection that delivers a representative subset of the data points. The mean-shift algorithm is more computationally expensive, as several seed points are required to find one of the clusters (k-means requires only one seed point per cluster). In the d-dimensional data space, every seed point is moved following the local gradient of point density in $\mathbb{R}^d$ in a stepwise manner, and finally most of the shifted seed points converge to a density maximum. Several points that meet at a density maximum vote for one cluster, the location of the density maximum in $\mathbb{R}^d$ is called a *mode*.

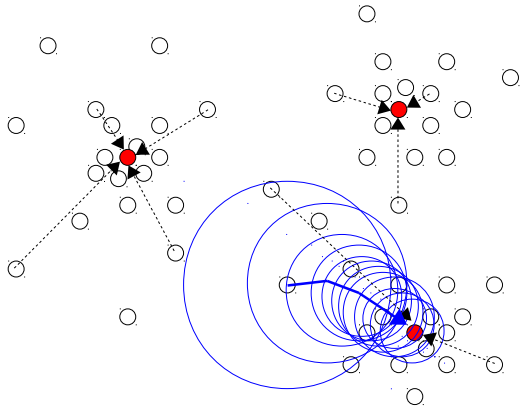

Figure 5: *Adaptive mean shift*

Figure 5 shows the functioning of the *adaptive mean-shift* procedure. The basics of adaptive meanshift are explained in [CRM01]. The blue arrow to the lower right cluster points to the mode of this cluster (red point). It marks a location of maximum density. The arrow starts at a data vector that is chosen as one of the representative sample points and lies in a region that is comparatively sparse. Sparse regions may be far away from the associated cluster center (mode). Furthermore, representative information about the density in sparse regions can only be gained if a large hypersphere is queried. The query range depends on the size of the *filter kernel* (blue circles in figure 5) that is used to select the local neighborhood. Hence, sparse regions require a larger kernel in order to constrain the variance of the estimated densities during the shift procedure and the shift vector should have a large magnitude. The magnitude of the shift vector is adjusted according to the local density, otherwise the procedure might overshoot the mode.

Thus, the adaptive mean-shift procedure allows for faster convergence than a fixed-bandwidth mean-shift. Therefore, for cluster detection/mode finding I chose the *Fast Adaptive Mean Shift* algorithm that is described in [GSM03] and implemented in C++.

3.2.4. Edges and umbilic points

Non-smooth meshes such as architectural models frequently contain vertices on orthogonal edges or plane surfaces (umbilic points). Both are special cases with respect to the surface descriptor based on principal curvatures. Umbilic points are removed by the first selection step (*3.2.2 Procedure*), so they are not available for transformation calculation. Points on orthogonal edges or corners are problematic because the normals and curvatures are estimated completely wrong.

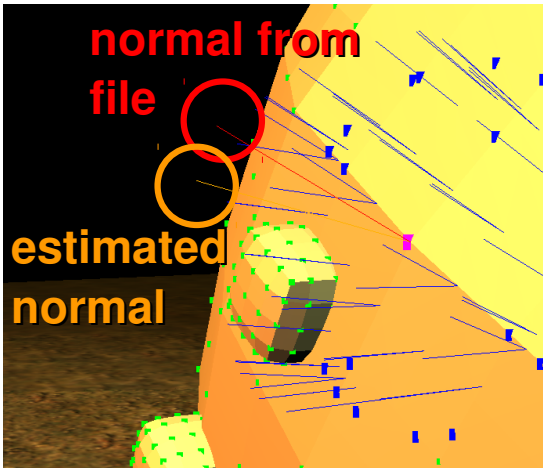

Figure 6: *Normal estimation: The normal vector is a byproduct of curvature estimation*

Figure 6 shows characteristics of the curvature estimation method described by [ACSD*03]. A sphere around the vertex is regarded. The angles between the blue lines are the angles between the face normals of adjacent triangles. Together with the vectors (orientations) of the edges within the sphere they influence the resulting curvature tensor. The principal curvature orientations as well as the normal vector (orange) are eigenvectors of the curvature tensor, the eigenvalues correspond to their magnitudes.

At corner points, the curvature tensor is not calculated as expected (figure 7). The same applies for vertices on flat surface areas (figure 8). This need not be problematic, since for similar vertices the tensor is similar anyhow. But a more robust solution for non-smooth meshes are *edge signatures*:

Edge signatures. A way out of the problem is to use a signature calculation method that is suitable for the particular form of architectural meshes. It is not necessary nor sensible to calculate accurate curvatures at edge vertices. Rather the centers of edges can be regarded instead of vertices. The smaller principal curvature κ_1 of an edge's center is always 0. However, the greater principal curvature κ_2 is either 0 (edges on a flat surface) or greater than 0. If the curvature is greater than 0, it cannot actually be calculated, but it can be replaced by the angle between the adjacent triangle surfaces with normals $\mathbf{n}_a$ and $\mathbf{n}_b$, respectively, of the edge with orientation and length defined by the vector $\mathbf{e}$ (equation 4, $\mathbf{e}$ is used in equation 5). These curvature replacements, k_1 and k_2, can be used instead of real curvatures.

$$k_1 = \kappa_1 = 0 \tag{3}$$

$$k_2 = \cos^{-1}\mathbf{n}_a \cdot \mathbf{n}_b \tag{4}$$

$$k_2 = (\cos^{-1}\mathbf{n}_a \cdot \mathbf{n}_b)|\mathbf{e}| \tag{5}$$

Equation 5 includes the length of the edge $|\mathbf{e}|$ in order to enhance the uniqueness of k_2 and therefore the signature.

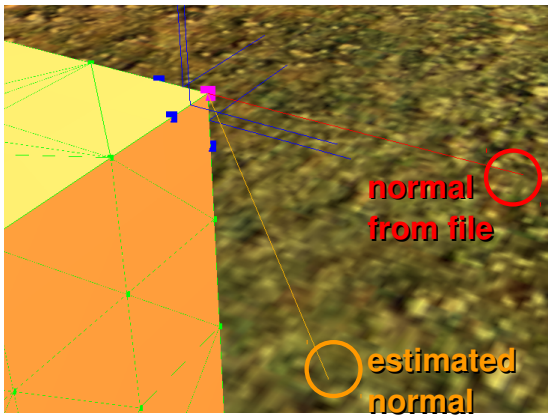

Figure 7: *Non-smooth surface, corner: The estimated normal vector points into a completely different direction than the corner tip*

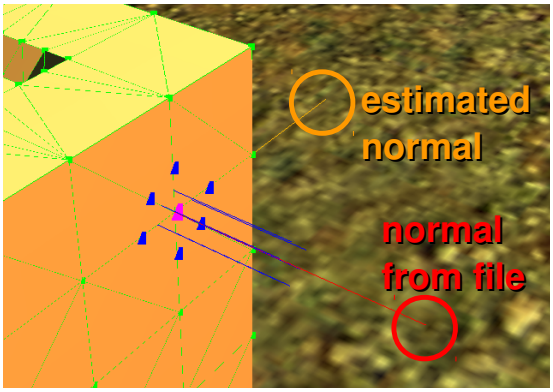

Figure 8: *Flat area*

3.2.5. From modes to structure elements

In order to locate and select entire structure elements, the modes resulting from the clustering procedure (vectors in $\mathbb{R}^7$) are used to find the transformation vectors of the cluster. Every transformation vector is associated with the pair of points it links. The nearest neighbors of a mode in transformation space are searched until a certain *distance threshold* is exceeded. The threshold can be adjusted by regarding the distances of the already accumulated neighbors.

The *first points* $\mathbf{p}_{\mathbf{T}_k,1}$ of the point pairs of the accumulated neighboring transformations $\mathbf{T}_k$ are points/vertices of one instance of a structure element, just as the *second points* $\mathbf{p}_{\mathbf{T}_k,2}$ are points/vertices of another instance.

3.3. Results

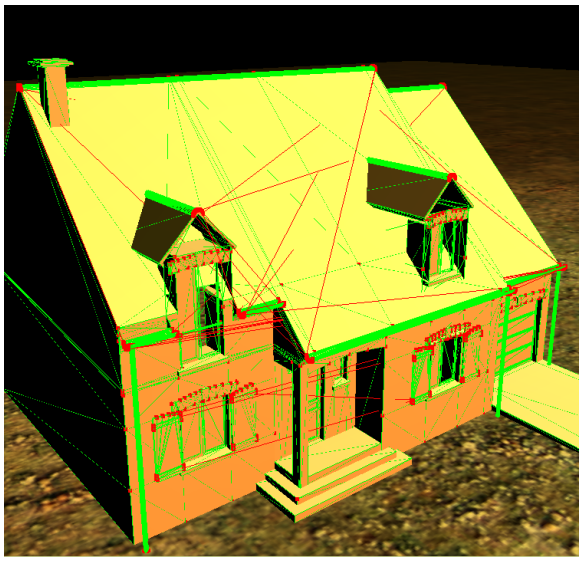

Figure 9: *Curvature-based approach: transformations*

Figure 9 shows the transformations that are nearest to the modes. They identify large-scale symmetry relations between the ends of the rain pipes as well as between the window shutters.

4. Framework

The research was conducted using a home-brewed testing application called *SemHouse* alias "Dr. House", that is a combination of the *Ogre3D* framework and a *.NET*-based GUI window. The *.NET* framework provides the programmer with the ability to combine the convenient but bytecode-based *C#* programming language for GUI development and less expensive algorithmic tasks with fast machine code procedures written in C++. The bridge between C# and mere C++ is provided by the *C++/CLI* language which extends the C++ language by .NET elements and therefore allows to produce and interact with .NET objects.

4.1. Ogre3D

Ogre3D is a 3D engine that features high-level access to low-level graphics application programming interfaces such as OpenGL or DirectX. It provides the programmer with basic $\mathbb{R}^3$ navigation (keyboard and mouse) and a simple model loader that is capable of loading Ogre *MESH* files that contain geometric models. The *MESH* format can be transformed into an XML file which allows the programmer to examine its contents in a text editor.

The link between Ogre3D and .NET is called *Mogre*, which is an Ogre3D wrapper written in C#. Mogre is available in the form of a dynamic-link library for Visual Studio .NET. Standard Ogre3D functions such as geometric transforms or navigation can be conveniently accessed through .NET-style methods and properties that are supplied by Mogre, but the access to geometric elements such as vertices or edges is not intended and requires the use of unsafe blocks of C# code.

4.2. C++/CLI

The C++/CLI language binds the C++ language to the .NET common language infrastructure. Performance of C++ programs remains the same as with a pure C++ compiler, but there is one drawback: The link between .NET and C++ is not transparent, so function parameters cannot be passed by reference or pointer but rather need to be duplicated. This affects speed and memory consumption and therefore reduces the adoption of C++/CLI to C++ algorithms with moderate parameter size and comparatively costly tasks such as frequent nearest neighbor searches after passing the vectors to build the search structure or performing the *Fast Adaptive Mean Shift* clustering as described above (*3.2.3 Mean-shift clustering*).

4.3. User interface

Figure 10 shows the controls rack of the testing application, which can be arbitrarily expanded by adding new controls if required. The selected tab ("Meanshift-based") shows parameters for the *curvature-based approach*. The *Fast Adaptive Mean Shift* algorithm requires the parameters K, L for *locality-sensitive hashing* and the parameter k for *pilot density estimation* (number of neighbors). They are conservatively preset and can be reduced in order to gain optimal performance (please refer to [GSM03]).

5. Conclusion

I introduced two different approaches to detect symmetry relations in architectural models. Both of them can be used for detecting structure elements that are instanced at least two times on a monolithic mesh. Depending on the geometric characteristics of a structure element, such as the vertex count or the uniqueness of the local surface patches around its vertices, detection will succeed or fail. Further research might be geared towards regarding a model's textures as an additional source of information for symmetry detection.

6. Acknowledgements

I thank Michael Wimmer for his support, particularly in the early stages of my work.

References

[ACSD*03] ALLIEZ P., COHEN-STEINER D., DEVILLERS O., LÉVY B., DESBRUN M.: Anisotropic polygonal remeshing. *ACM Trans. Graph. 22*, 3 (2003), 485–493.

[AMAK07] ATHANASIADIS T., MYLONAS P., AVRITHIS Y., KOLLIAS S.: Semantic image segmentation and object labeling. *IEEE Transactions on Circuits and Systems for Video Technology 17*, 3 (March 2007), 298 – 312.

[CMM02] COMANICIU D., MEER P., MEMBER S.: Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24* (2002), 603–619.

[CRM01] COMANICIU D., RAMESH V., MEER P.: The variable bandwidth mean shift and data-driven scale selection. In *in Proc. 8th Intl. Conf. on Computer Vision* (2001), pp. 438–445.

[GSM03] GEORGESCU B., SHIMSHONI I., MEER P.: Mean shift based clustering in high dimensions: A texture classification example. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision* (Washington, DC, USA, 2003), IEEE Computer Society, p. 456.

[Mer07] MERRELL P.: Example-based model synthesis. In *In I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (2007), ACM Press, pp. 105–112.

[MGP06] MITRA N. J., GUIBAS L. J., PAULY M.: Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph. 25*, 3 (2006), 560–568.

[WMWG09] WEBER B., MÜLLER P., WONKA P., GROSS M.: Interactive geometric simulation of 4d cities. *EUROGRAPHICS 2009 28*, 2 (2009).

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), ACM, pp. 669–677.

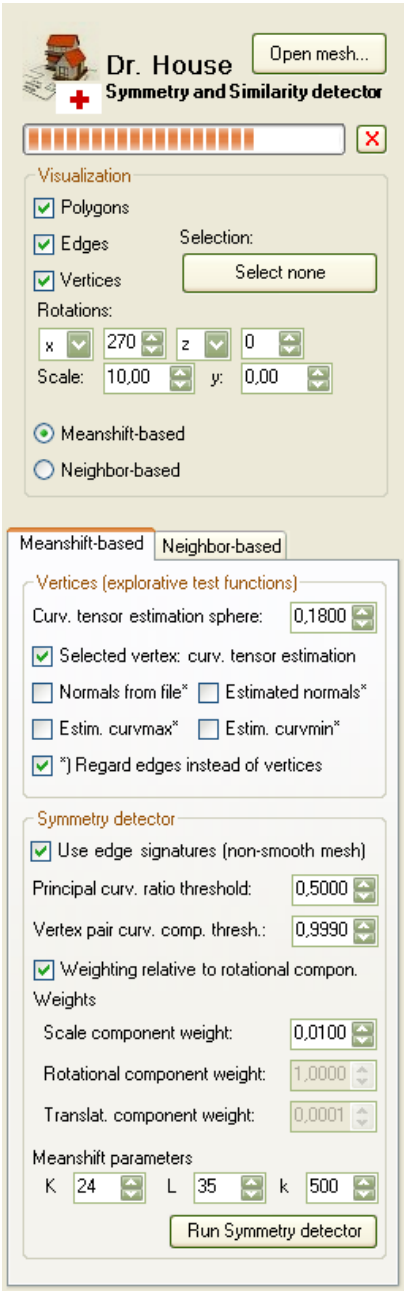

Figure 10: *Controls rack*