Dynamic Label placement for Improved Interactive Exploration

Thierry Stein, Xavier Décoret

Dynamic Label Placement for Improved Interactive Exploration

Thierry Stein*
Grenoble Universities/INRIA

Xavier Décoret†
Phoenix Interactive

Figure 1: *An example of dynamic labeling for a complex scene. See accompanying video for better demonstration.*

Abstract

This work presents a novel approach for dynamically rendering annotations attached to a 3D scene. We formulate the problem as a general optimization under constraints, accounting for certain desirable properties. To approximately solve the NP-hard optimization problem in real-time, we present a particular heuristic that greedily places labels while maintaining constraints. Typical greedy label placement algorithms do not pay particular attention to the order of placement and, as a result, suffer from the fundamental limitation that successive labels get progressively more difficult to place. We use algorithmic and mathematical tools that compensate for the drawback of typical greedy approaches. In addition, they are well suited for GPU implementation, because they are completely image based. As a result, we can place tens of labels in real-time, as demonstrated in this paper.

CR Categories: I.3.6 [Methodology and Techniques]: Interaction techniques—Labelling

Keywords: Labelling, Apollonius diagram, SAT, GPU

*Thierry.Stein@imag.fr
†x.decoret@phoenix-i.fr

1 Introduction

One striking characteristic of humans is the ability to communicate rich and complex information. We have invented several abstractions to communicate complex information simply— writing and drawing being natural examples. It is impressive indeed, that many tools available for expressing information today were, at best, works of science fiction decades ago. As Hanrahan [Hanrahan 2005] pointed out at his keynote speech at Eurographics 2005, charts, pictorial graphs, diagrams, and spreadsheets are examples of such tools.

The challenge when creating representations for communication is that of *abstraction* and *expressiveness*. While the former enables simplicity in communication by accounting for context, the latter allows emphasis of certain ideas over others. The sheer bulk of data that is available from more recent inventions such as the internet have posed yet another challenge. Classification, organization, indexes, comments and annotation are essential elements of presenting what would otherwise be merely raw data, as information.

Of the many recent tools for effective communication, our interest lies in 3D representations. Expressing information through 3D visualization has become popular for a wide gamut of applications ranging from entertainment (video games e.g. World Of Warcraft, Quake, etc) to serious applications (geolocalization, virtual tours, cartography, medical visualization, etc). In addition, 3D graphical communication has created a new form of social interaction (e.g. Second Life). However, in most of these applications, the 3D visualization alone is generally insufficient to effectively and comprehensively convey the information. While the 3D representation is useful for presenting a view and helps immersion and localization

it is often overwhelming in terms of the amount of information conveyed. A key aspect in this communication with 3D representations is labeling, or visibly attaching text that provides more information about certain regions. Labeling has been a standard tool that enables attaching information and also drawing attention to certain regions. (e.g. life points of other players in WoW, name of avatars in Second Life, description of shops in Google Maps and indicating organ dimensions in medical applications).

Figure 2: *World of Warcraft uses world-placed billboards to overlay labels onto a 3D view, which causes cluttering when too many objects are close in screen space* ©.

While labeling has been extensively used in 2D illustration, it is indeed challenging to extend and automate the labeling process for images of 3D representations. Figure 2 shows a typically cluttered view in WoW where the text indications enormously simplify quick comprehension. In this paper, we tackle the problem of displaying such information more optimally, which relates to *labeling* as reviewed in Section 2.

The key point here is that the viewpoint is "inside" the scene. A consequence of being immersed is that label placement poses more challenges than when labels have to be layed out "around" an annotated object, as typically the case in most existing work.

Rather than to produce aestethic (i.e close to what talented illustrators can do) layouts for particulars view, our goal is to guarantee that all annotations relevant for the current viewpoint are adequately presented to the user, while attempting to minimize interferance with the 3D experience of scene navigation. In particular, we pay attention to enforce temporal coherence.

2 Related Work

This work is mainly related to the one of *map labeling*, where algorithms try to place annotations around points, lines or regions. Finding an optimal solution to this problem has been proven to be NP-hard [Marks and Shieber 1991]. The map labeling problem has been extensively studied in the static 2D case. A good survey of the most efficient solutions can be found in [Christensen et al. 1995]. Recently, [Daiches 2006] adressed the problem of zooming in the map with consistent labeling.

External labeling is introduced by [Fekete and Plaisant 1999] as an alternative to internal labeling for dense sets of points. Labels are placed on the border of a circle containing anchor points. Their algorithm is limited to the 2D case. A good formalization of the static 2D case is found in [Bekos et al. 2005]. Labels are positioned around an axis-aligned bounding box, and the length of leader lines is minimized. The major limitation is that labels need to have a common size. In [Vollick et al. 2007] the problem is cast as a minimization of an energy function. The motivation was to capture style from a user-specified example, and reuse it to generate labels on novel inputs. They use simulated annealing for resolution, which needs a few minutes, and the approach is consequently non interactive.

An early mention of 3D external labeling is found in [Preim et al. 1997] where the authors present an algorithm for coherent zooming. Additionaly, [Strothotte 1998] explain the problem of temporal coherence, and show two simple cases where there is no solution to prevent popping for a rotation. In [Bell et al. 2001], authors present an algorithm for real-time 3D label placement using a view-management strategy introduced in [Bell and Feiner 2000]. The scene is approximated by a set of bounding boxes, and labels are interatively placed in the nearest empty rectangle. They also introduce hysteresis to minimize temporal discontinuity. [Hartmann et al. 2004] model the problem using potential fields. Labels are positionned by diffusion of an energy, starting from the middle of the scene. However, this algorithm does not manage all the constraints, and in particular, cannot prevent leader lines intersections. They solve it by permutation of labels, which is not always valid for labels of different sizes. [Ali et al. 2005] present a flexible pipeline including different style. They are also limited to single line text and they don't manage objects with holes. Finally, [Götzelmann et al. 2007] present a solution for animated 3D models. The main restriction is that they need to analyze the entire animation, so it is not applicable to interactive navigation.

3 Problem formulation

We start with a 3D scene with points of interest tagged with semantic information. The user navigates the scene, and the information is rendered overlayed on the 2D view. As an example, imagine a virtual model of a museum gallery exposing statues. The pedestal of each statue is tagged with its name, and the name of the artist that committed it. On the *Venus de Milo*, the shoulders are tagged with an explanation on why the arms are missing. On the *David*'s head, a tag indicates that proportions are not anatomically correct, but chosen so that they appear correct when seen from ground level. When the user enters the gallery, all statues are visible, and only the names are shown. When she moves closer to one statue, the related information appears gradually. This example scenario is what we call *dynamic labeling*. The term "label" is ambiguous. It refers both to the semantic information (e.g. the *text* explaining what a point of interest indicates), and to the rendering of that information (e.g. the *marks* that are overlayed on the 2D view to display the text with a particular font and alignment). Let's first clarify the terminology we use.

We call *label* the semantic information attached to a point of interest, which is called the label's *anchor*. The anchor is a 3D point in the scene. The semantic information must contain a displayable *content*, but it can also contain other information such as a category id, an importance, etc. For a given view of the scene, a label's anchor projects to a given 2D position that we call *anchor point*. The label is rendered inside a 2D axis-aligned box that we call the *label box*. The position and size of this box is what is to be determined by our system. However, the aspect ratio of the box is fixed by the content. We do not consider dynamically layed out content such as [Jacobs et al. 2004]. We instead assume the content is given as an "image" of given proportions by the user. Consequently, the size of a label box can only controlled by a *scale* factor. Finally, we call *leader line* the line that connects the anchor point to the center of the label box. Figure 3 summarizes the terminology. We will

also use the following notations, where uppercase indicates input values while lowercase indicates output values, to be computed :

$\mathbf{A}_i = X_i, Y_i$	2D coordinates of anchor point i
$\mathbf{d}_i = W_i, H_i$	width and height of label content i
$\mathbf{a}_i = x_i, y_i$	2D coordinates of the center of label box i
$[\mathbf{a}_i, \mathbf{A}_i]$	Leader line of label i
s_i	scale factor applied to label content ; the label box has 2D dimensions $s_i W_i \times s_i H_i$

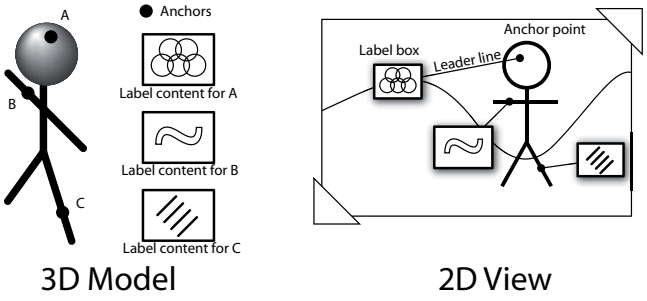

Figure 3: *Terminology used in dynamic labeling*

The problem is thus as follows. Given a set of n labels'contents and associated anchor points, how to best overlay label boxes onto the scene view. For that, we must define what "best" means. We choose the following criteria. Label boxes must not intersect each other, or some information would not be visible. Leader lines must neither intersect other leader lines nor label boxes, or the anchor/label relation would not be immediately perceived. The label boxes should not hinder the navigation: they should hide as less as possible of the 3D scene, and they should move coherently over time. The anchor/label relation should be as direct as possible: leader lines should not be too long, so that the anchor point and the label box are visually closed; and leader lines should be oriented along priviledged directions. Finally, the label boxes scales should be as close to 1 as possible.

The two first criteria are hard constraints ("must"). The last ones are recommendations ("should"). Consequently, we model the problem as an minimization under constraints. We search over the space of all possible $\{x_i, y_i, s_i\}_{i=1\ldots n}$ such that:

$$\forall i \neq j \;\; [\mathbf{a}_i, \mathbf{A}_i] \cap [\mathbf{a}_j, \mathbf{A}_j] = \{\} \quad (1)$$

$$B(\mathbf{a}_i, s_i\mathbf{d}_i) \cap [\mathbf{a}_j, \mathbf{A}_j] = \{\} \quad (2)$$

$$B(\mathbf{a}_i, s_i\mathbf{d}_i) \cap B(\mathbf{a}_j, s_j\mathbf{d}_j) = \{\} \quad (3)$$

where B is an axis-aligned 2D rectangle specified by its center and its diagonal vector. We minimize an energy function:

$$\begin{aligned} E(\{x_i, y_i, s_i\}) = \alpha \sum_i |\mathbf{a}_i\mathbf{A}_i| \quad (4) \\ + \beta \sum_i |1 - s_i| \\ + \gamma \sum_i |\cos\theta_i| + |\sin\theta_i| \\ + \delta \sum_i \iint_{B(\mathbf{a}_i, s_i\mathbf{d}_i)} f(\mathbf{x}) d\mathbf{x} \end{aligned}$$

where f is a positive real function defined (in image space) over the current view, and whose role is explained in a moment. The first term controls the length of leader lines. The second controls that the labels are not scaled to 0. The third term is used to favor leader lines that are horizontal or vertical, θ_i being the angle that the leader line i does with the horizontal axis. The fourth term is used to mesure how much the labels occludes the 3D scene underneath. The function f is called *occupancy function*, and is obtained as follow. Each surface in the scene is attached a user-specified scalar value in $[0, 1]$ that indicates how important it is that this surface be not occluded by labels. For example, we would put 0 on walls, and 1 on objects. This importance value can be view-dependent, for example decreasing with the depth value of pixels. We render a view of the scene from the current viewpoint using this value as surfaces's colors. The resulting image gives f. By integrating the value of f over the label boxes, we have a measure of how the scene is occluded by labels. By using this in the energy function, instead of adding it as a constraint, we allow the algorithm to place labels that hide some parts of the scene when it is not possible to find a better placement. Yet, it will strive to occlude parts that are less important first.

4 An Extended Greedy Algorithm

Minimizing eq.(4) under constraints eq.(1-3) is a complex task (NP-hard) that cannot currently be done in real time. Our approach is to solve it greedily. For each label whose anchor point is visible in the current view, we choose (x_i, y_i, s_i) so that the contribution to E is minimum, and so that the constraints are respected. This approach does not yield the exact minimum, because the ordering of labels matters. Moreover, this approach has in principal two drawbacks. First, even if we reduce the dimension of the search space from $3n$ to 3, it is still a large space to explore. Second, as common with greedy algorithm, if we do not order labels adequately, we may end up with the last labels highly increasing E, or worst, being impossible to place without violating the constraints, which would require backtracking the previously placed labels. Figure 4-top/right shows such a poor placement. We address these two problems as follows.

We reduce the search space by removing s_i from the minimization. We simply fix the scale of label box i based on the distance to the observer. This is a very crude simplification of our general formulation, and the reader may argue that we should have omitted it in the first place. Yet, we believe that adjusting the scale gives a lot of flexibility to find an ideal label placement. Hence a general formulation should include it, even if the solution presented in this paper cannot yet take it into account. Indeed, reducing the dimension of the search space to 2 is the key to interactivity. This allows to use the GPU to very efficiently evaluate the cost of each candidate (x_i, y_i), and then retrieve the minimum through matrix reduction, as shown in Section 5.

Further, we carefully chose the ordering of labels so that we first place those that are *a priori* hard, leaving the easiest ones for the end. Hopefully, and we observe it in practice, this should help to alleviate the inherent drawback of the greedy approach mentioned above. Intuitively, the hardest labels are those for which there are few candidates (x, y) that meet the constraints, or only candidates for which the contribution to E would be high. We identify two situations that generate such labels. The first situation is when the anchor point is surrounded by other anchor points. If we place the outer labels first, there will be no room left for the inner one. The second situation is when the anchor point is "deep inside" an occupied region, that is, the closest point $\mathbf{x}$ for which $f(\mathbf{x})$ is small is not that close. In other words, the leader line will be long.

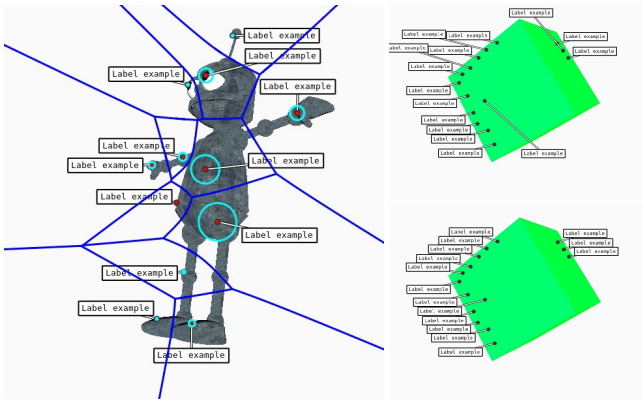

Figure 4: *(left) Example of Apollonius diagram. Note that boundaries of the cells are curved. (right) result of greedy placement with random ordering of labels (top) and our ordering (bottom).*

4.1 Ordering the labels

Here is a way we could solve the first problem alone. We compute a Voronoi diagram of the anchor points. We then "peel" layers of anchor points from the "outside" to the "inside". More mathematically, we take all unbounded cells of the diagram. By property of Voronoi, each such cell has exactly two adjacent unbounded cells. This yields a circular ordering of the cells. We push the corresponding anchor points in that order in a LIFO queue. We then recompute the Voronoi diagram of the remaining anchor points and repeat the process until no anchor point is left. Popping the queue would now traverse the labels from the most surrounded one to the less surrounded one.

Here is now a way to solve the second problem alone. Starting with the function f, we compute a distance field. That is, for each $\mathbf{x}$, we find the distance to the closest pixel at which f is 0. Such a distance field can be computed very efficiently on GPU as shown in Section 5. We now evaluate the distance field at each anchor point and sort in decreasing order.

The two solutions above yield different orderings and have complementary advantages and drawbacks. Consequently, we need a way to somehow merge them into one. Our solution is to use the peeling algorithm described above, but with an *Apollonius diagram*, also known as *Additively weighted Voronoi diagram* [Karavelas and Yvinec 2002]. The Apollonius diagram of a set of *site* points c_i with weight w_i is a subdivision of the plane into connected regions, called *cells*, associated with the sites. As for Voronoi diagram, the cell of a site (c_i, w_i) is the locus of points on the plane that are closer to c_i than any other c_j. But this time, the distance of a point x in the plane to a site c_i is defined as $|x - c_i| - w_i$. In other words, within a cell, the most important point around is the associated site. The weight for each point is set to the value of the distance function. In other words, each anchor point is "considered" a disc tangent to the closest silouhette, and we compute the Voronoi diagram of these discs. Figure 4-left shows such a diagram and the resulting ordering of anchor points.

4.2 Evaluating contribution

Now that the anchor points are ordered, we traverse them and for each label i, we consider all possible locations (x, y) for the label box (there is a finite number of pixels in an image so we can enumerate these locations). At each location, we test if the label box can be placed there without violating the constraints. This can be done in one operation using morphological operators, as now described. We traverse all previously placed labels j. For each, we compute the shadow region of the label box j and leader line j for a point light source at anchor point i. This gives us the location where there would be intersection between the leader line of label i and other labels. We then perform a dilatation by the size of the label box i, which is straighforward knowing the vertices of the shadow region. We finally render all these regions in an offscreen texture as a binary image. The resulting binary image contains 0 where we can place label box i without violating constraints, and 1 elsewhere. Figure 5 illustrates the process. There are two important points here. First, the constraint checking information is available as a texture. Second, the use of dilation allows to test validity using one texture lookup. Both properties allows to integrate the constraint testing gracefully into a GPU implementation based on matrix reduction as described in Section 5. Once this is

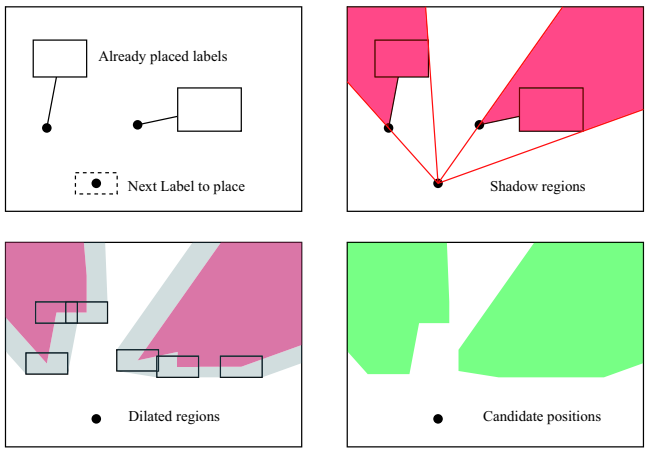

Figure 5: *Testing constraints against already placed labels using morphological dilatation.*

checked, we evaluate the contribution to E that would result from placing the label box there. One interesting aspect of this approach, besides making the test atomic, is that we can prevent label boxes to be too close from each other by dilating by a larger rectangle than the actual label box.

Looking at eq. (4), we see that we have to compute an integral with the scene, which is potentially costly. To do it efficiently, we compute a *Summed Area Table* (SAT) of the scene. SAT has been originally introduced by [Crow 1984] as an alternative to mipmapping for texture filtering. It allows to compute the integral on an axis-aligned boxes with only four arithmetic operations. Recently, [Hensley et al. 2005] has presented a fast algorithm to compute SAT on the GPU. Once again, if we want a "band" between objects silhouettes and label boxes, we can integrate on enlarged domains.

4.3 Managing temporal continuity

As mentioned earlier, we want the displayed label not to hinder the navigation experience. The user should not be distracted by popping or fast-moving labels. Intuitively, the label box placement should be continuous, that is, if the user changes slightly the viewpoint, the label box should be only slightly translated. The algorithm we just described does not account for temporal coherence, because the optimisation only involves the current viewpoint. To address this, we add an extra term to the cost function, that penalizes a label's position that would be too far from the label's previous position. This favors continuity whenever a solution exists.

Unfortunately, there are cases where continuity is not achievable. As shown in [Strothotte 1998], there are discontinuities inherent to the problem. Figure 6 shows another example.

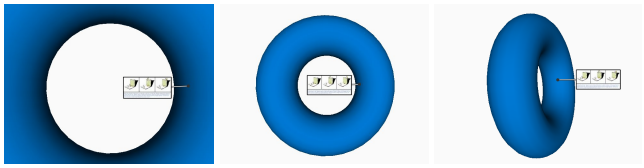

Figure 6: *Placement of labels can not be continuous. The three images above shows three consecutive views (zooming out, then rotating) of a torus with one label attached. In the first view, the label must be placed inside the torus. In the last view, it cannot. There will necessary be a time when the label needs to be "teleported"*

To address this problem, we interpolate between the previous position of a label box, and its next position. That is, each label box continuously tries to reach the position it has been placed too, but its velocity is bounded, and it may need several frames to reach this position. Fundamentally, the problem is not solved and the label is actually "teleported", but the process "smoothes" the teleportation to avoid visual discontinuity. It serves only as a last resort, when the configuration has no continuous solution.

Another source of discontinuities comes from the treatment of invisible anchor points. As mentioned in first paragraph of Section 4, we only place labels for those anchor points that are visible. But what happens when the visibility of an anchor point changes, for example from visible to hidden? At the very moment before the anchor point becomes hidden, the label box it still completely visible. If placement were continuous, at the moment after the anchor point becomes hidden, the label box would be still largely visible. If we simply remove the label when the anchors becomes hidden, we will have a visual discontinuity because the anchor point is punctual (with a null area) while the label box is not. This problem is well known in NPR when attaching strokes to seed points on a mesh[Vanderhaeghe et al. 2007].

Our solution to that problem is to make the point non-punctual. That is, we test the visibility of a small 3D sphere around the anchor point. Using occlusion queries, we can count how much of this sphere is visible, and we use that to blend the label box with the background. In other words, we no longer have a binary visibility status, but a continuously varying one. This has the extra benefit of avoiding the precision issue that would occur when testing the visibility of a single point. The drawback, however, is that in certain situations, such as an anchor seen through fencing or through folliage, the label will sort of "blink" with alternate fade-in and fade-out.

5 Implementation and Results

Previous section described our approach and its motivation. In this section, we provide the details of implementation, and in particular how well our approach maps to the GPU. The steps are:

1. Render occupancy function
2. Compute a Summed Area Table and distance transform on it
3. Compute order with Apollonius graph
4. For each label in that order
   (a) Traverse all pixels in image
   (b) If label box cannot be placed at that pixel, skip to next pixel, otherwise evaluate the contribution if label box was placed at this pixel
   (c) Find the pixel with minimum contribution using matrix reduction.

All these computations are performed on the GPU, typically rendering a full screen quad with a specific fragment shader, in order to transform an offscreen texture to another offscreen texture.

The key point is that there is very limited transfer to the CPU. Indeed, we never need to transfer back the whole offscreen textures. We compute Summed Area Table and Distance Transform with the same single fragment shader using a jump flooding algorithm [Rong and Tan 2006]. Step 3 requires the distance field only at the anchor points. Step 4-b can perform all the constraint tests and cost function evaluation within the fragment shader because SAT and dilated mask make those atomic operations. Finally, step 4-c is done using matrix reduction, which requires only $\log n$ passes where n is the dimension of the offscreen textures, and a final readback of one pixel (which contains in the R channel the minimum function, and in the GB channels the coordinates at which it occurs).

Another keypoint is that the offscreen texture resolution is completely decorellated from the resolution of the observer's window. Its size only influences the number of (x, y) samples considered to place the label boxes. Since the cost of our algorithm is directly proportional to the resolution (through fill rate and matrix reduction passes), we can scale the quality to the power of the graphic cards. Note however that too small a resolution will cause flickering during the navigation.

We implemented this approach using non-optimized C++ and OpenGL code. Our tests were ran on a GeForce 7800, using an offscreen resolution of 512×512. We are able to interactively place up to 20 labels at 30Hz (but the scene itself can contains much more labels, if not all of them have visible anchor points). Note that the number of polygons in the 3D scene is not relevant for performance measurements. Indeed, we need to render the scene once to obtain the occupancy function. Every subsequent step is entirely in image space and no longer depends on the geometric complexity of the scene, but only on the offscreen resolution chosen.

The user can control globally the weighting coefficients $\alpha, \ldots, \gamma$ of eq.(4). In the future, the values of those coefficients could be specified per label. This would allow to force certain labels to be vertical/horizontal, while letting other free. The user can also tag the surfaces with an importance value. In our demos, we have simply split the scene in two files, one with objects of importance 0 (typically walls) and one with objects of importance 1 (typically furniture). Figure 7 shows some results. The accompanying video shows how our algorithm behaves dynamically.

Our approach does have limitations. The main one is inherent to the greedy approach. The last labels may be placed very far from their anchor point, or in the worst case, may not be placed. Even if our ordering heuristic greatly reduces this problem, there are typical cases that fail for our algorithm. Because we peel anchor points from the outside to the inside, the heuristic will be inefficient in the case shown by Figure 8-left, where anchor points should intuitively be peeled in the inverse order. Another bad configuration is that of an extremely dense set of points. Even if we try to respect a circular order in label insertion, we can have an anchor point that is blocked between two previously placed labels. Finally, for large number of labels, there may not exist a valid solution, unless we also optimize over the sizes of the labels, which we do not handle for the moment.

6 Discussion and Future Work

We have presented a general formalization of dynamic labeling of interactive scenes, together with a pratical greedy approach to solve the NP-hard optimization problem. Our main contribution is to use Apollonius diagrams to select labels in an appropriate order and compensate the drawback of the greedy approach. Our second contribution is an efficient implementation on the GPU, through the use of SAT and morphological operations to make the evaluation of constraints and costs atomic operations. As can be seen in the video, this allows interactive navigation of 3D scenes with annotations.

Our formulation is very general, even though we do not exploit it entirely for the moment. In the future, we would like to optimize also the sizes of the labels. We would also like to test offline optimization algorithms and compare the solution found with the approximate one found by our heuristic. Finally, we would like to investigate recent advances in greedy approaches such as GRASP [Cravo et al. 2008] to improve optimization without sacrificing performance.

Acknowledgments

We thank Laurence Boissieux for 3d models and Alexandrina Orzan, Kartic Subr and Pierre-Edouard Landes for their help with regard to video editing and english reviewing, and for their moral support during the last few days.

References

ALI, K., HARTMANN, K., AND STROTHOTTE, T. 2005. Label Layout for Interactive 3D Illustrations. *Journal of the WSCG 13*, 1, 1–8.

BEKOS, M. A., KAUFMANN, M., SYMVONIS, A., AND WOLFF, A. 2005. Boundary labeling: Models and efficient algorithms for rectangular maps. In *Proc. 12th Int. Symposium on Graph Drawing (GD'04)*, Springer, J. Pach, Ed., vol. 3383, 49–59.

BELL, B. A., AND FEINER, S. K. 2000. Dynamic space management for user interfaces. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 239–248.

BELL, B., FEINER, S., AND HÖLLERER, T. 2001. View management for virtual and augmented reality. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 101–110.

CHRISTENSEN, J., MARKS, J., AND SHIEBER, S. 1995. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graph. 14*, 3, 203–232.

CRAVO, G. L., RIBEIRO, G. M., AND LORENA, L. A. N. 2008. A greedy randomized adaptive search procedure for the point-feature cartographic label placement. *Comput. Geosci. 34*, 4, 373–386.

CROW, F. C. 1984. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 207–212.

DAICHES, E. 2006. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics 12*, 5, 773–780. Member-Ken Been and Senior Member-Chee Yap.

FEKETE, J.-D., AND PLAISANT, C. 1999. Excentric labeling: dynamic neighborhood labeling for data visualization. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, 512–519.

GÖTZELMANN, T., HARTMANN, K., AND STROTHOTTE, T. 2007. Annotation of animated 3d objects. In *SimVis'07*, SCS Publishing House e.V., T. Schulze, B. Preim, and H. Schumann, Eds., 209–222.

HANRAHAN, P. 2005. Realistic or abstract imagery: The future of computer graphics.

HARTMANN, K., ALI, K., AND STROTHOTTE, T. 2004. Floating labels: Applying dynamic potential fields for label layout. In *Smart Graphics*, 101–113.

HENSLEY, J., SCHEUERMANN, T., COOMBE, G., SINGH, M., AND LASTRA, A. 2005. Fast summed-area table generation and its applications. *Computer Graphics Forum 24*, 3, 547–555.

JACOBS, C., LI, W., SCHRIER, E., BARGERON, D., AND SALESIN, D. 2004. Adaptive document layout. *Commun. ACM 47*, 8, 60–66.

KARAVELAS, M. I., AND YVINEC, M. 2002. Dynamic additively weighted voronoi diagrams in 2d. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, Springer-Verlag, London, UK, 586–598.

MARKS, J., AND SHIEBER, S. 1991. The computational complexity of cartographic label placement. Tech. Rep. TR-05-91.

PREIM, B., RAAB, A., AND STROTHOTTE, T. 1997. Coherent zooming of illustrations with 3d-graphics and text. In *Proceedings of the conference on Graphics interface '97*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 105–113.

RONG, G., AND TAN, T.-S. 2006. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 109–116.

STROTHOTTE, T. 1998. *Computational Visualization: Graphics, Abstraction, and Interactivity*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

VANDERHAEGHE, D., BARLA, P., THOLLOT, J., AND SILLION, F. 2007. Dynamic point distribution for stroke-based rendering. In *Eurographics Symposium on Rendering, Rendering Techniques 2007, May, 2007*, A K Peters Ltd, Grenoble, France, J. Kautz and S. Pattanaik, Eds., 139–146.

VOLLICK, I., VOGEL, D., AGRAWALA, M., AND HERTZMANN, A. 2007. Specifying label layout style by example. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 221–230.

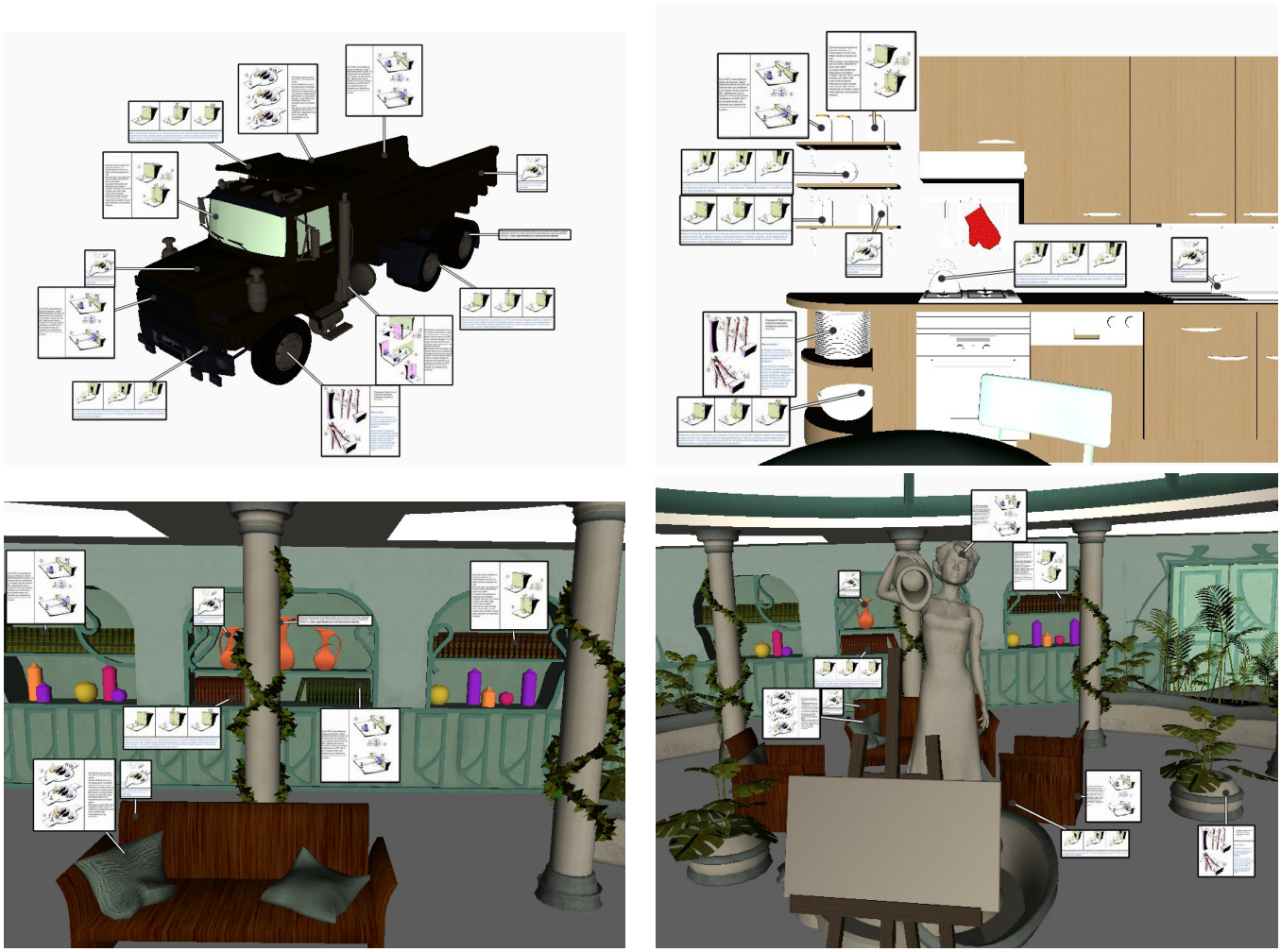

Figure 7: *Some results obtain with our method*

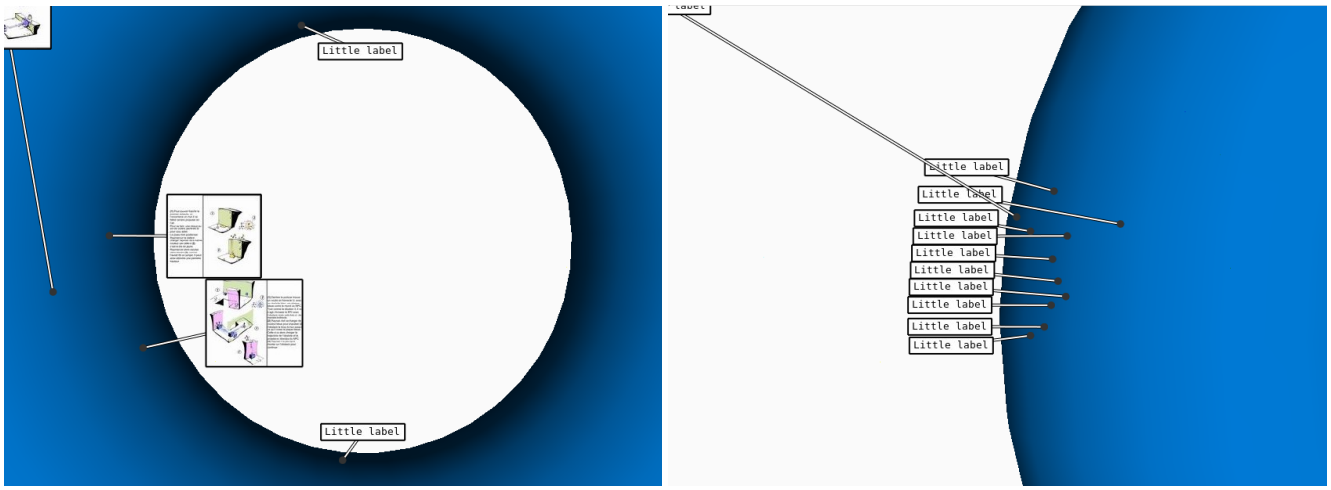

Figure 8: *When our algorithm fails to place a label, it is placed in the upper corner. Here are two cases where this may happen (left) the ordering heuristic is designed to place labels "around" and object, not "inside" a hole, for which the peeling order should be reversed. (right) dense set of labels quickly saturate free space and we can no longer place an anchor line.*