Shape Depiction for Transparent Objects with Bucketed k-Buffer

David Murray, Jérôme Baril, Xavier Granier

▶ **To cite this version:**

David Murray, Jérôme Baril, Xavier Granier. Shape Depiction for Transparent Objects with Bucketed k-Buffer. Eurographics Symposium on Rendering - Experimental Ideas & Implementations, Jun 2016, Dublin, Ireland. pp.33-39, 10.2312/sre.20161207 . hal-01338535

HAL Id: hal-01338535
https://inria.hal.science/hal-01338535v1

Submitted on 28 Jun 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2016), pp. 1–7
E. Eisemann and E. Fiume (Editors)

Shape Depiction for Transparent Objects with Bucketed k-Buffer

D. Murray[1,2] and J. Baril[2] and X. Granier[1]

[1]LP2N (UMR 5298 CNRS, Univ Bordeaux, IOGS) - Inria, Talence, France [2]FEI Bordeaux, France

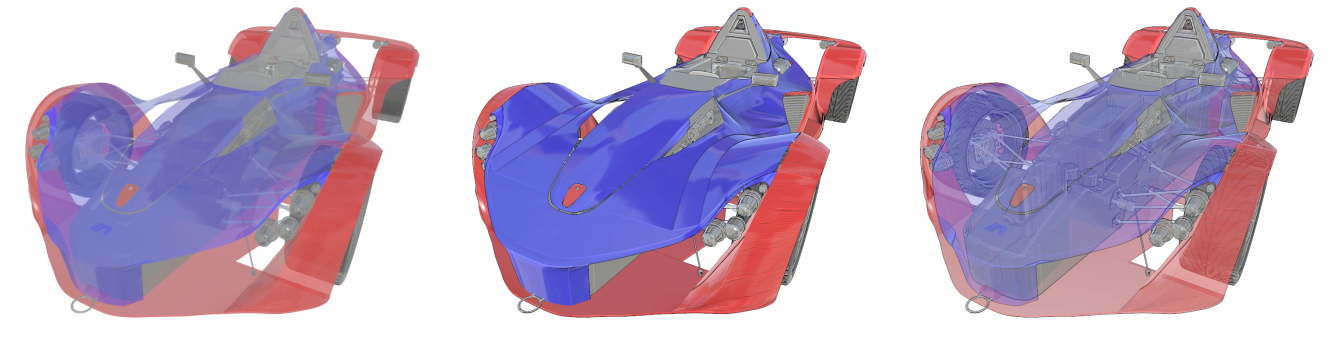


Figure 1: *Enhancing surface with Mean Curvature Transparency and Bucketed k-Buffer. On the left of the image is classical rendering (OIT) with constant transparency. In the center is shape depiction on opaque surfaces, while on the right is our method (using the transfer function prensented in Figure 7(a)). Observe how object details on each surface is highlighted to provide a better understanding of the composition of the object. The model runs on our system at 64 FPS with OIT and 24 FPS with our solution, for a resolution of 1000 by 1000 pixels and 16 layers of transparency. Model courtesy of dessindus.blogspot.fr.*

Abstract

Shading techniques are useful to deliver a better understanding of object shapes. When transparent objects are involved, depicting the shape characteristics of each surface is even more relevant. In this paper, we propose a method for rendering transparent scenes or objects using classical tools for shape depiction in real time. Our method provides an efficient way to compute screen space curvature on transparent objects by using a novel screen space representation of a scene derived from Order Independent Transparency techniques. Moreover, we propose a customizable stylization that modulates the transparency per fragment, according to its curvature and its depth, which can be adapted for various kinds of applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

The simulation of a lighting model for visualizing three-dimensional scenes is an essential tool for understanding the content of the scene. While current real-time rendering techniques are convincing for everyday objects, some models suffer from lack of real landmark and thus, from grasp of the shape. Actually, transparency is a common tool for analysing complex object subtleties, e.g. Computer-Aided Design. However, with raster-based applications, ambiguities from using classical alpha-blending are two-fold: (i) a correct composition involves to render the scene back-to-front; (ii) composition of multiple non-related fragments introduces a bias in shape perception. The back-to-front composition issue (i) is solved by Order-Independant Transparency (OIT) methods. However, the second one (ii) has no obvious solution.

In this paper, we provide customizable tools that help to work around shape perception ambiguities during alpha-blending. Our predicate is that shape perception relates on expressive rendering methods. Among the large set of techniques available in this domain, we are particulary interested in shape depiction algorithms that aim at stylizing the rendering of a shape according to its features. Basically, these features can be classified into three categories: surface orientation, surface curvature and inflection points. While the first one can be considered as supplied during rendering process, other ones should be computed during the rendering in

order to benefit from screen-space formulation. Screen-space approaches for feature extraction have proven useful for exploring dense meshes obtained by photogrammetry using Radiance Scaling [GVP*12], with applications in Cultural Heritage. We consider this as we focus on the exploration of complex models (e.g. CAD), which require efficiently computing curvature at a scale adapted to the current viewpoint. The computation of surface properties is linked to derivative considerations: curvature is a derivative of the normals and we need a derivative of the curvature to find inflection points. Using discrete differential geometry operators, the derivative computation in screen-space involves to have access to the neighborhood of a point, i.e. a fragment located at a pixel position. For opaque object, depth-test operation is enough to eliminate unnecessary fragments. Thus, with one fragment per pixel, finding neighbor information is straightforward. For transparent objects, a pixel may contain multiple fragments. Since a differential operator must be applied per fragment to perform shape depiction locally, the complexity of using such an operator is overweighted by the complexity of finding neighbor information.

Thus, our main contribution is two-fold. First, we introduce the Bucketed k-Buffer, a data structure suited to find fragment's neighborhood efficiently. Second, we present a customizable stylisation that modulates shading and transparency per fragment, according to its curvature and its depth, which can be adapted for various kinds of application.

2. Related Work

Shape depiction techniques aim at conveying a message based on shape properties. Some previous methods propose to modify the reflection pattern according to constraints such as geometric features, material properties or stylisation. *Gooch Shading* [GGSC98] relies on a cool to warm color transition based on normal consideration giving information about surface orientation. With the same idea, Sloan et al. [SMGG01] propose to use precomputed *Lit Sphere* addressed with surface normals. Barla et al. [BTM06] extend this idea by adding depth information to mimic aerial perspective. *Exaggerated Shading* [RBD06] uses normals at multiple scales to enhance the relief of a shape. Using higher order consideration, Kindlmann et al. [KWTM03] modulate the shading between concavities and convexities based on object-space curvature. Vergne et al. [VPB*09] propose to compute image-space curvature to shift the normals of a surface, thus exaggerating the reflection patterns. The screen-space approach has the additional benefit of exaggerating curvature magnitudes toward object contours, which increases surface legibility. Later, Vergne et al. [VPB*10] propose a generalisation of this technique for any model of illumination and any type of material.

Line-based rendering have also been used to convey geometric informations. Initiated by Saito and Takahashi [ST90], line-based techniques may be used to depict purely intrinsic geometric properties, for instance ridges and valleys (e.g. [OBS04]) or surface inflections (e.g. [KST08]). It can also enhance view-dependent features like occluding contours, *Suggestive Contours* [DFRS03] and later *Suggestive Highlight* [DR07], or *Apparent Ridges* [JDA07], an view-dependent proposition of [OBS04].

Most previous techniques work on opaque surfaces with raster-based applications. Dealing with transparent surfaces involves to consider *Alpha Blending* issues, solved by Order-Independent Transparency techniques. Prior is *Depth Peeling* [Eve01], which consists in peeling the scene by doing several opaque geometry passes, while stripping away the layers already rendered at each pass, enhanced later by Bavoil et al. [BM08]. With the availability of atomic operations on GPU, Yang et al. [YHGT10] propose an implementation of the *A-Buffer* [Car84], a per-pixel linked list of fragments constructed in a single geometry pass and sorted by depth before blending. In the same way, for a fixed number of fragment per pixel, the *k-Buffer* [BCL*07], stores only the k nearest fragments. Dual pass *k-Buffer* [Kub14] is a variant that uncorrelates depth and color information. Finally, McGuire et al. [MM16] relies on a commutative blending function which uses weights applied on transparency to approximate correct blending to simulate physical phenomena.

Combining OIT with expressive rendering methods is addressed by Hummel *et al.* [HGH*10], with Normal-Variation Transparency. However, as they use built-in GLSL function, neighborhood is inacurate. Carnecky *et al.* [CFM*13] proposed an altered *A-Buffer*, in which fragments are connected to their neighboring fragments. According to the connexity and the logical depth of a fragment, transparency is modulated such that creases are emphasized to nearly opaque, smoothly decreasing to nearly transparent otherwise. Günther *et al.* [GSE*14] propose an enhancement with surface patches to make part of a surface transparent if an object is behind. However, to our knowledge, there is no method that computes screen-space geometric features per fragment to modulate shading and transparency in real-time. Based on a tailored data structure that allows such computation, our method provides a way to address existing shape depiction techniques on transparency shapes.

3. Solution overview

The goal of the presented method is to enhance shape depiction for transparent object. For such intention, our process is composed of three main steps as shown in Figure 2:

1. Scene discretization into a screen-space representation.
2. Per fragment feature-based stylization.
3. Blending of the generated fragments.

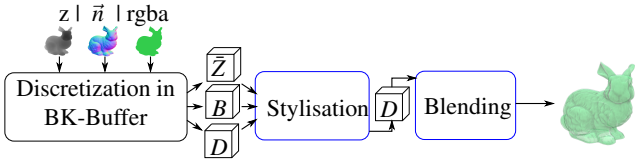


Figure 2: *Pipeline used for feature-based stylization with Bucketed K-Buffer. The discretization pass fills the buffer Z, the bucket buffer B and the data buffer D with their respective information (see Section 4). The stylization extracts features and applies modulation on fragment's data (see Section 5). Finally, the blending pass consists in traditional OIT back-to-front composition.*

The first step consists in capturing the scene into the bucketed k-buffer (BKB). The BKB is a discretization of the scene such that fragments are organized pixelwise into three 3D data structures: Z, B and D. The roles of Z and D are nearly the same as data structures used in dual pass k-buffer [Kub14] to capture, respectively, depth

and data. B is used to group fragments belonging to the same shape per bucket. Details of the BKB's data structures are presented in Section 4. Note that the BKB contains geometric and color data per fragment. The second step is a stylization process which extracts geometric features from the BKB (i.e. based on the normals) to enhance shape depiction (i.e. by modulating shaded color). The stylization is customizable by providing user-defined transfer functions indexed by the fragment's depth and curvature (see Section 5). The last step consists in reordering fragments back-to-front in order to realize a correct alpha-blending.

Every stage of our system is performed in real-time on modern graphics hardware. Results are presented in Section 6.

4. Bucketed K-Buffer for efficient neighbor query

Leaving aside commutative OIT which excludes per fragment operations, we focus on per pixel fragment list. We seek a way to efficiently access a fragment's neighborhood. First, we introduce the concept behind our solution, that is, the elaboration of the Bucketed k-Buffer. Then, we provide several practical details about the method.

4.1. Definition of the structure

Our structure relies on a k-Buffer basis. Indeed, with current GPU, *Depth Peeling* is obsolete. *A-Buffer* relies on a linked list which implies unbound memory. Moreover, all fragment information is stored in the same buffer. For a better flexibility on data structure, the dual pass *k-Buffer* [Kub14] technique is well-suited. Indeed, by decorraleting depth and color, we can manipulate only depth or color information separately. Furthermore, storing only the k nearest fragments in k layers ensures a total control over memory occupancy. Those two points are key elements for the choice of our structure.

In such structure, a naive approach to find the best neighbor candidate is to iterate through each neighboring pixel's fragment list. However, this implies costly memory access. Vardis et al. [VVP16] use buckets to decompose a fragment list such that fragments are grouped by depth range. In the same way, we propose to reorganize the fragment lists so that fragments belonging to a same object are regrouped in a bucket. That way, only fragments in the corresponding bucket needs to be traversed, thus avoiding many memory access. Using such partition, finding a one-ring neighborhood would go down, in theory, from a complexity of $O(k^2)$ per pixel to $O(kn)$, where k is the number of fragments per pixel and n is the number of fragments in a bucket, so we should have a gain as soon as $n < k$.

The proposed algorithm shares the same concepts as the dual-pass *k-Buffer* in the sense that we use two geometry passes to store depth and data in separate buffers. However, in order to facilitate access to neighbors, our data structures differ in the data we store and we use additionnal computing passes to arrange the data according to our needs. The remainder of this section describes the pipeline to build the BKB data structure in a sequential order. In the following, W and H denote respectively the width and the height of the rendering window.

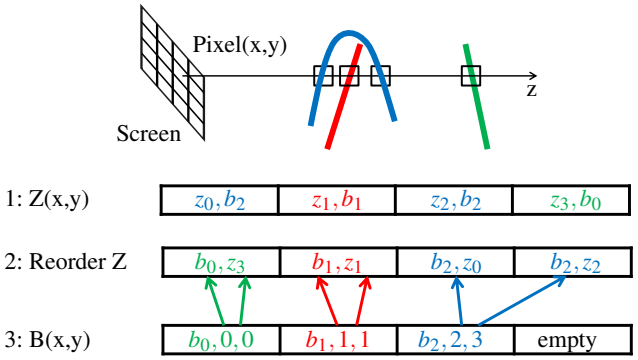


Figure 3: *Description of the discretization process. In the first pass (1), Z is filled with a depth-ordered fragment list, containing depth (z) and shape ID (b) for each fragment. In the second one (2) Z is reordered by shape Id. Then (3), B is filled with bucket's information : shape Id (b), first and last fragment index (i, j).*

Z insertion Such as the dual pass *k-Buffer* [Kub14], we store fragment's depth per pixel in the first geometry pass. Relying on the benefit of such approach, the n nearest fragments from the camera, $n \leq k$, are stored and ordered by depth in the buffer Z. For each fragment, we keep not only its depth (z) but also its shape ID (b) (see Figure 3 (1)), where b is a integer representing the identifier of the shape where the fragment is from. Thus, the Z data structure is of size $W \times H \times k$, containing pairs (z, b).

Z reordering During this pass, we reorder per pixel fragment pairs of the Z data structure along b instead of z (see Figure 3 (2)). Sorting fragment pairs in this manner allows grouping fragments that belong to the same shape consecutively in order to gather them into buckets.

Bucket creation Per pixel groups of fragments, which share the same shape ID, are implicitly defined by the reordering of Z in the previous pass. However, in order to have a direct access to the range of these groups, the B data structure is built in a computing pass. Such groups are defined as buckets: a bucket contains the shape ID (b, i.e. a bucket ID) and the indices (i, j) of the first and last fragments of a group. The bucket buffer B contains $W \times H \times k$ triplets (b, i, j) (see Figure 3 (3)).

Since n fragments are stored in Z during the first pass, $n \leq k$, no more than k buckets are stored per pixel. However, storing bucket information in a table indexed by b can lead to indices that are out of range if $b > k$. Thus, B is considered as a per pixel hash table in which each bucket is indexed by $\mathcal{H}(b)$, where $\mathcal{H}$ is a hash function defined as:

$$\mathcal{H}(b) = b \mod k$$

Using such hash function may lead to collision in the hash table. Thus, during insertion of a bucket b in buffer B, we check that the slot at index $\mathcal{H}(b)$ in the hash table is free. If the slot is unoccupied, the bucket information is stored at this index. If the slot is occupied, b is incremented until the index $\mathcal{H}(b)$ points to a free slot. Note that a free slot is always available since the number of buckets per pixel cannot be greater than k, the size of the hash table. The same

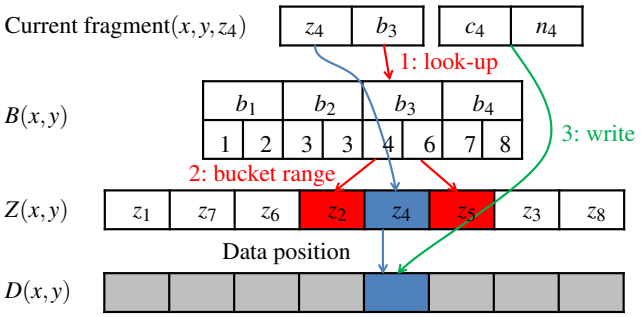


Figure 4: *Fragment query for data storage pass. For a generated fragment, the shape ID is used to locate, with a linear search, the associated bucket (1) and extract its range (2). The current fragment is located within these range. To find its index, a depth comparison is used. Fragment's data, color and normal, are stored at this index (3). For neighbor query, the fragment (x,y,z) is used for a look-up in $B(x+dx,y+dy)$ and $Z(x+dx,y+dy)$.*

incremental process is used to find a bucket, except that we check if the slot contains the right bucket. Organized in that way, B, in combination with Z, is used as an index lookup to perform efficient fragment query as shown in Figure 4.

Data storage To store fragment's data, a second geometry pass is performed to fill the buffer D. The fragment's data are its color (c) and normal (n). Only fragments pre-selected during the first geometric pass must be stored. Thus, for each generated fragment, we perform a fragment query as shown in Figure 4. If the query is successful, data are stored to the given index, or rejected otherwise. Buffer D thus contains $W \times H \times k$ pairs (c,n).

4.2. Implementation details

As presented in the previous section, we manipulate three 3D structures. Each of them will be able to contain $W \times H \times k$ elements, where W and H correspond to the viewport size and k is the number of layers in the buffer.

During the first pass, the sorted-insertion by depth is done using atomic comparison. As atomic comparisons rely on a bitwise AND operator, depth must be used as Most Significant Bit, even if we store both depth and shape number. These two values are packed on 32 bits with 16 bits each, in the buffer Z. Using depth as MSB ensures that we keep the k-nearest fragments.

For the construction of the buckets, depth and shape number are swaped inside Z. This way, shape number is now used as MSB instead of depth and we can easily reorder our list. After reordering, buckets are constructed as described in the previous section. Using this structure requires 32 bits per bucket : 8 bits for the first fragment position, 8 bits for the last one and 16 bits for the shape number.

Finally, to insert the data in the buffer D, we rely on a binary search done directly inside the correct bucket. This bucket is located using the process in Figure 4 with a condition on shape number. Once we know the correct fragment's layer, we can store the fragment color after shading to use only 32 bits for color. The shading relies on the Bidirectionnal Reflectance Distribution Function (BRDF) proposed by Disney [BS12] for direct illumination and Image-Based Lighting to approximate indirect illumination. We store as well the fragment's normal, packed on 32 bits (15 bits for each x and y components and 2 for the sign of z). This results in using 64 bits per fragment for the data structure.

5. Feature-driven stylisation

In this section, we demonstrate how to use the BKB for an efficient feature-driven stylization. First, we present how to find fragment's neighborhood to compute curvature per fragment. Then, an application of stylization is shown adapted from existing shape depiction techniques.

5.1. Feature extraction

Carnecky et al. [CFM*13] propose a method to find the neighbor of a fragment. A comparison is performed between a fragment and all the fragments contained in an adjacent pixel. The comparison function (ε) is based on normal and depth differences and the best neighbor candidate is the fragment which obtains the minimum value. We use a similar approach with the following criteria:

$$\begin{aligned} \varepsilon_n(i,j) &= 1-\vec{n_i}\cdot\vec{n_j} \\ \varepsilon_z(i,j) &= |z_i-z_j|^2 \\ \varepsilon(i,j) &= \varepsilon_n(i,j)+\varepsilon_z(i,j) \end{aligned}$$

To ensure that we correctly identify the neighborhood, each fragment labeled as the best candidate does a reverse check to determine if this candidate also views the current fragment as a neighbor. If one of the neighbors is not found, we use a special value indicating a missing neighbor. Figure 5 illustrates the interest of this reverse check. Comparing to the work of Carnecky et al. [CFM*13], a crucial point in our method is the lower complexity of this algorithm. Actually, thanks to the BKB, fragments belonging to the same object are grouped together. Thus, as explained in the previous section, the number of pairs to test is restricted to a subset of candidates, i.e. the ones with the same shape number. We access to this subset using the same process as data storage, presented in Figure 4, except that the query is done on the neighboring pixel's lists (buckets and fragments).

The neighbor information is used to compute screen-space curvature per fragment, such as Vergne et al. [VPB*09]. In this approach, principal curvatures, κ_1 and κ_2, are extracted from surface normals. Instead of directly using of the mean curvature $(\kappa_1+\kappa_2)/2$, we prefer, for practical reason, the following mapping varying between -1 and 1:

$$\kappa = tanh(\kappa_1+\kappa_2)$$

For the rest of this paper, κ is considered as a curvature value. In our convention, -1 corresponds to a convetixity et 1 to a concavitiy. Contrary to Vergne et al. [VPB*09], our method does not suffer from undefined behavior at the object's boundaries. By taking advantage of potential missing neighbors, we can handle those aera: the curvature of a fragment with a missing neighbor is set to $\kappa = -1$.

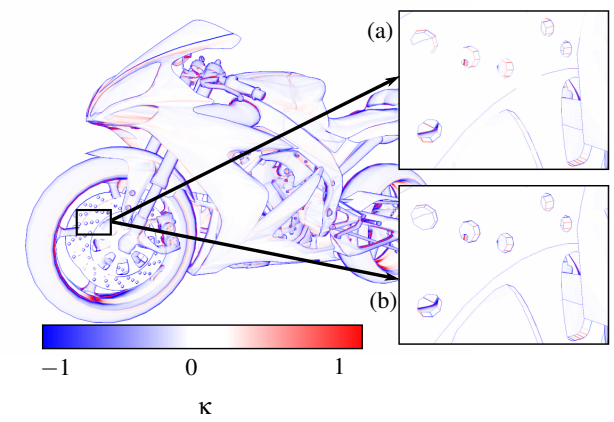


Figure 5: *Importance of doing a reverse check. (a) shows a simple check, which result in biased results at creases. (b) shows a reverse check, with creases identified as such.*

5.2. Stylization

For the stylization, our method modulates both shaded color and transparency. These modulations are controlled by transfer functions: an 1D texture for color and a 2D one for transparency.

The shaded color is modulated with a curvature-indexed transfer function, presented in Figure 6(a). This function is inspired by *Mean Curvature Shading* [KWTM03] and *Radiance Scaling* [VPB*10], with bright convexities and dark concavities. It provided convincing results for our application, as we wanted to modulate a physically-based shading. However, for other applications, a different transfer function could be used in our pipeline for other stylisations as shown in Figure 6(b).

The modulation of the transparency extends the work presented by Hummel *et al.* [HGH*10] as we use curvature instead of normal-variation considerations. Based on the idea of X-Toon [BTM06], we propose to modulate opacity with a transfer function indexed by absolute mean curvature and depth. A value of 1 corresponds to fully opaque and 0 to fully transparent. An example of transfer function can be seen in Figure 7. We choose to use the absolute mean curvature so that either convexities or concavities would have the same opacity. Indeed, based on our observations, both are crucial to fully grasp shape charactestics.

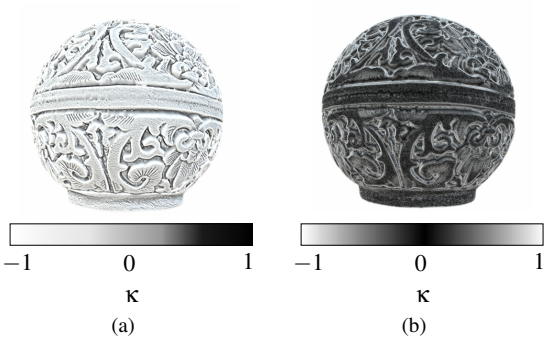


Figure 6: *(a) The transfer function we use to modulate shading brighten convexities and darken concavities. (b) Another example : bright curved surfaces, dark flat ones.*

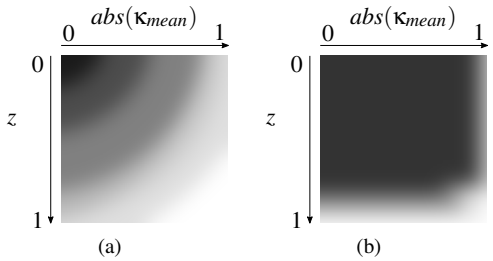


Figure 7: *Example of different transfer functions. White corresponds to opaque (opacity of* 1*) and black to transparent (opacity of* 0*). Both follow the same idea: more transparency on flat and close surfaces, more opacity on curved surfaces as well as far ones.*

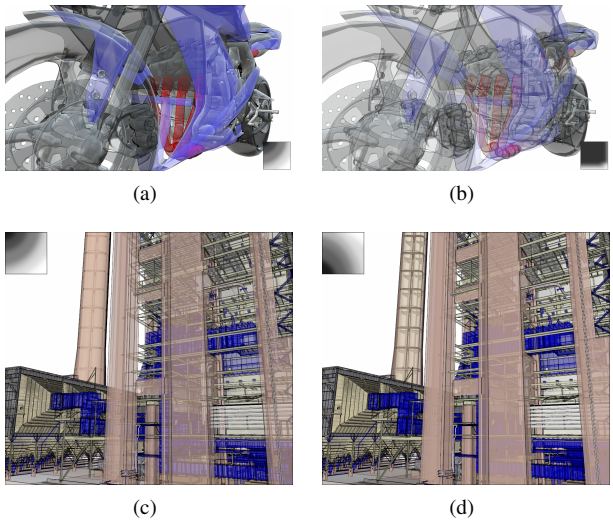
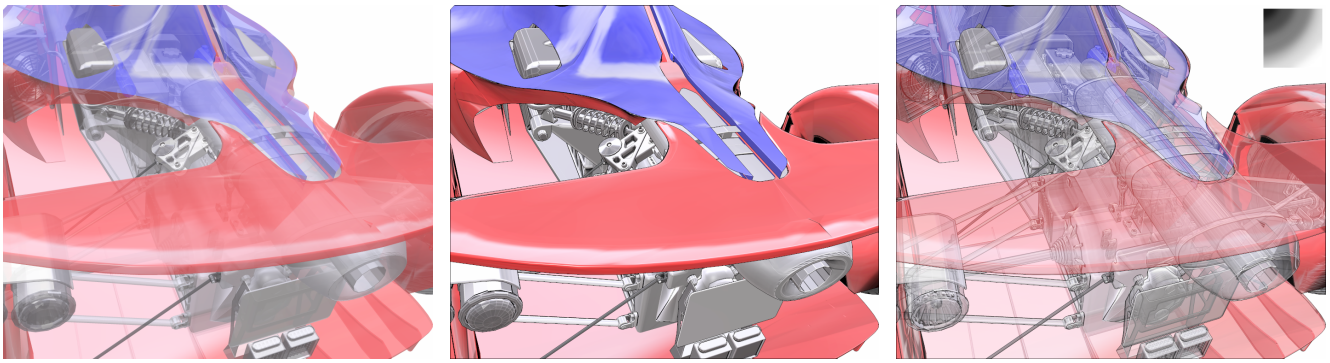


Figure 8: *Examples of Mean Curvature Transparency. (a,b) The transfer functions used are respectively the one presented in Figure 7 (45 FPS versus 120 without modulation). (c,d) The transfer functions are shown in the top left corner (10 FPS, versus 20 without modulation). Times are obtained with a resolution of 1200 by 800 for the bike 1000 by 1000 pixels for the UNC Powerplant and 16 layers.*

Our system allows the use of any transfer function to modulate opacity. Figure 8 illustrates how different functions allow to emphazise specific parts of the scene. For instance, the function used in (a) and (c) results in nearly transparent fragments on flat surfaces close the viewport, while curved surface are nearly opaque. The opacity quickly grows as the distance to the camera increases. This behavior allows to place more focus on the front part of the scene. On the contrary, the function in (b) places less focus in front but depicts shape on a wider range of depth. Finally, (d) presents another behavior, where front parts are more opaque than far ones. Both use the function presented in Figure 6(a) to modulate shading so that convexities are enlightened while concavities are darkened.

6. Results and performances

To illustrate the benefits of extending shape depiction to transparent surfaces, Figure 9 shows a comparison of: (a) simple OIT, (b) enhancing convexities and concavities on opaque surfaces and (c)



(a) (b) (c)

Figure 9: *(a) OIT using a k-Buffer with a constant opacity value of 0.6 (50 FPS). (b) Mean Curvature Shading only. (c) Mean curvature transparency (20 FPS). The transfer function used is on the top-right corner. Model courtesy of dessindus.blogspot.fr. Times are obtained with a resolution of 1000 by 1000 pixels and 16 layers.*

with transparency, enhanced as well with our method. OIT allows only to guess the internal structure of the car while Mean Curvature Shading enhances surface features only on the outer parts. On the contrary, our method enhances both internal and external parts, with a focus on the front of the scene.

To measure the impact on performances of our method, we use a synthetic test configuration, in which rendering time is only dependent of the number of objects. The scene is composed of 32 stacked full-screen quads so that each of the 32 layers are always filled. Each quad is located at a random depth, and is associated with a random object. We then vary the maximum number of object, as described in Figure 10. We compare the result of our solution with the brute force approach. Note that the brute-force approach has a constant rendering time, as complexity is only dependent on the number of fragments. The latter roughly corresponds to the method of Carnecky et al. [CFM*13]. Implementation and renderings were done using the Open Inventor® 9.6 SDK by FEI™ with an NVIDIA GTX 980. Results are shown on Figure 10 for 32 layers and a resolution of 512x512 (around 8,4 million of fragments). If our solution is obviously much slower that simple transparency (10 ms in this configuration), we can see that it improves performances for transparent shape depiction as soon as there is more than one bucket. With more than five buckets, we halve the time required, going down to a fifth of the performances of the brute force approach when there are many objects.

These observations fit rather well with theorical gain of Section 4, going from $O(k^2)$ to $O(nk)$, but only when n is at least 2. Indeed, when $n \leq 2$, the cost of accessing all the fragments in a bucket becomes nearly equal to the cost of accessing the bucket, whereas for $n > 2$, the bucket cost becomes quickly insignificant with regards to the cost of accessing all fragments. However, this gain is valid only for neighbor query, as our structure still requires two sorting pass. In our algorithm, we use insertion sorting which has a complexity of $O(k^2)$ in the worst case, $O(k)$ when fragments are nearly-sorted, which is often the situation with our structure.

The last aspect concerns memory consumption. The proposed solution requires up to 128 bits per fragment (32 bits for depth and

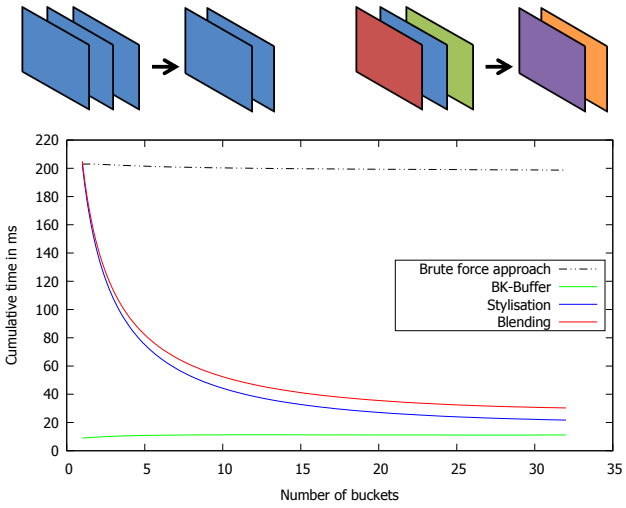


Figure 10: *Cumulative rendering times of a fixed number of fragments (32) for different bucket configurations (colored quads) and 512 by 512 pixels. Full lines depict cumulative times of our method, the dot line represents the total time of the brute force approach. As a reference, a dual pass k-Buffer alone with constant transparency requires around 10 ms per frame.*

bucket id, 64 bits for color and normal, and 32 for bucket informations) whereas a classic dual pass k-Buffer requires 64 bits per fragment. For a Full HD resolution, this over is around 32 MByte per layer versus 16 MByte for a simple k-Buffer.

7. Conclusions and limitations

We have presented a new approach to propose one solution to the bias in shape perception introduced by compositing multiple non-related fragments. We convey geometric properties of transparent surfaces with Mean Curvature Transparency based on the Bucketed k-Buffer. The BKB allows an easy access to the fragments of a shape. Moreover, we provide a customizable tool to enhance scene perception for transparent surfaces. Based on a user-defined transfer function, it is possible to choose which features should be

highlighted. The choice is highly dependent of which features need to be enhanced, as well as the complexity of the scene.

Our approach is dedicated to scenes with multiple objects: on a unique object, such an approach introduces too much complexity. Our structure performs better than existing ones, even if it implies a overhead in memory. However, our data structure may be optimized for more applications than neighbor query. In particular, we could use it with editing techniques like Fast-Edit, or extend the amount of optical phenomena that we currently simulate, like refraction, sub-surface scattering or translucency with a limited impact on performances.

On the other side, in a scene with many small objects, results can become less legible. The current stylization is based on the feedback provided by our team. A further user-study, with a representative panel, would be necessary to accurately validate the perceptual gain of our method. In future work, we also plan to investigate the impact of using line-based rendering ([DFRS03, OBS04, JDA07, KST08], introduced in Section 2) to enhance third order features on transparent surfaces.

8. Acknowledgements

We thank FEI™ for funding this work and providing a licence of Open Inventor®, and the members of the Open Inventor® team for their support and feedback, particularly Fedy Abi-Chahla for his useful advices on OIT. We are grateful to the GrabCAD community (Car prototype and Suzuki R1, credits go to their respective designer) and the AIM@SHAPE VISIONAIR repository for the Circular box model.

References

[BCL*07] BAVOIL L., CALLAHAN S. P., LEFOHN A., COMBA J. L., SILVA C. T.: Multi-fragment effects on the gpu using the k-buffer. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (2007), ACM, pp. 97–104. 2

[BM08] BAVOIL L., MYERS K.: Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK* (2008), 1–12. 2

[BS12] BURLEY B., STUDIOS W. D. A.: Physically-based shading at disney. In *ACM SIGGRAPH* (2012), pp. 1–7. 4

[BTM06] BARLA P., THOLLOT J., MARKOSIAN L.: X-Toon: An extended toon shader. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR'06)* (Annecy, France, June 2006), DeCarlo D., Markosian L., (Eds.), ACM. 2, 5

[Car84] CARPENTER L.: The a -buffer, an antialiased hidden surface method. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1984), SIGGRAPH '84, ACM, pp. 103–108. 2

[CFM*13] CARNECKY R., FUCHS R., MEHL S., JANG Y., PEIKERT R.: Smart transparency for illustrative visualization of complex flow surfaces. *Visualization and Computer Graphics, IEEE Transactions on 19*, 5 (2013), 838–851. 2, 4, 6

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3 (jul 2003), 848–855. 2, 7

[DR07] DECARLO D., RUSINKIEWICZ S.: Highlight lines for conveying shape. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)* (Aug. 2007). 2

[Eve01] EVERITT C.: Interactive order-independent transparency. *White paper, nVIDIA 2*, 6 (2001), 7. 2

[GGSC98] GOOCH A., GOOCH B., SHIRLEY P., COHEN E.: A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), ACM, pp. 447–452. 2

[GSE*14] GÜNTHER T., SCHULZE M., ESTURO J. M., RÖSSL C., THEISEL H.: Opacity optimization for surfaces. 11–20. 2

[GVP*12] GRANIER X., VERGNE R., PACANOWSKI R., BARLA P., REUTER P.: Enhancing surface features with the radiance scaling meshlab plugin. In *Computer Applications and Quantitative Methods in Archaeology (CAA) 2012* (Southampton, United Kingdom, Mar. 2012), Chrysanthi A., Wheatley D., Romanowska I., Papadopoulos C., Murrieta-Flores P., Sly T., Earl G., Verhagen P., (Eds.), vol. 2 of *Archaeology in the Digital Era*, Amsterdam University Press, pp. 417–421. 2

[HGH*10] HUMMEL M., GARTH C., HAMANN B., HAGEN H., JOY K. I.: Iris: Illustrative rendering for integral surfaces. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (2010), 1319–1328. 2, 5

[JDA07] JUDD T., DURAND F., ADELSON E.: Apparent ridges for line drawing. In *ACM SIGGRAPH 2007 Papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. 2, 7

[KST08] KOLOMENKIN M., SHIMSHONI I., TAL A.: Demarcating curves for shape illustration. In *ACM Transactions on Graphics (TOG)* (2008), vol. 27, ACM, p. 157. 2, 7

[Kub14] KUBISCH C.: Order independent transparency in opengl 4.x. In *GPU Technology Conference (GTC)* (2014). 2, 3

[KWTM03] KINDLMANN G., WHITAKER R., TASDIZEN T., MÖLLER T.: Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Visualization, 2003. VIS 2003. IEEE* (2003), IEEE, pp. 513–520. 2, 5

[MM16] MCGUIRE M., MARA M.: A phenomenological scattering model for order-independent transparency. In *I3D 2016* (February 2016), p. 10. 2

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Ridge-valley lines on meshes via implicit surface fitting. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 609–612. 2, 7

[RBD06] RUSINKIEWICZ S., BURNS M., DECARLO D.: Exaggerated shading for depicting shape and detail. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, ACM, pp. 1199–1205. 2

[SMGG01] SLOAN P.-P. J., MARTIN W., GOOCH A., GOOCH B.: The lit sphere: A model for capturing npr shading from art. In *Graphics interface* (2001), vol. 2001, Citeseer, pp. 143–150. 2

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1990), SIGGRAPH '90, ACM, pp. 197–206. 2

[VPB*09] VERGNE R., PACANOWSKI R., BARLA P., GRANIER X., SCHLICK C.: Light Warping for Enhanced Surface Depiction. *ACM Transaction on Graphics (Proceedings of SIGGRAPH 2009) 28*, 3 (2009). 2, 4

[VPB*10] VERGNE R., PACANOWSKI R., BARLA P., GRANIER X., SCHLICK C.: Radiance Scaling for Versatile Surface Enhancement. In *I3D '10: Proc. symposium on Interactive 3D graphics and games* (Boston, United States, Feb. 2010), ACM. 2, 5

[VVP16] VARDIS K., VASILAKIS A. A., PAPAIOANNOU G.: A multi-view and multilayer approach for interactive ray tracing. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2016), I3D '16, ACM, pp. 171–178. 3

[YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 1297–1304. 2