Adaptive Ray-Tracing on Spatial Patches

Victor S. Lempitsky, Denis V. Ivanov, Yevgeniy P. Kuzmin
Department of Mathematics and Mechanics, Moscow State University
Moscow, Russia

Abstract

Spatial patch is a new kind of rendering primitive that allows convenient and compact representation of surfaces for real-life 3D objects. In [1] and [2], several algorithms for z-buffer visualization of spatial patches are presented. However, z-buffering approach cannot provide photo-realistic quality of rendering. On the contrary, such quality can be obtained by means of ray tracing – a well-known method of realistic image synthesis. Being quite distinct from most of traditional ray tracing primitives, spatial patches possess different challenging features, which can be efficiently exploited by the modification of this technique. In order to exploit them, we propose to find parameters of ray - surface intersection approximately, where precision of operation is naturally defined by a size of a pixel. In case of spatial patches, such an assumption gives significant speed-up and possibility of rendering with optimal level of details.

In this paper, an efficient algorithm for ray-patch and ray-patched scene intersections calculation is presented. The paper starts with a method of representing spatial patches as hierarchies of bounding boxes. Employing existing techniques, these hierarchies can be linked into a single one that represents the whole scene. Then, the calculation of ray-tree approximate intersection is discussed. Experimental results, proving ray-tracing suitability of spatial patches, are presented as well.

Keywords: *Spatial Patch, Ray Tracing.*

1. INTRODUCTION

1.1 What is Ray Tracing?

Creation of photo-realistic images of 3D environments is a traditional application of computer graphics. *Ray tracing* is a powerful, simple, and the most widely used technique of this field. While more powerful approaches emerged in recent years, almost all of them employ ray tracing.

Image synthesis in classical ray tracing is performed as follows. A ray, starting in the observer eye and passing through the center of the pixel on the screen plane, is shot. Its intersection with the environment is found, determining an object visible through that pixel. Thus, a *primary* ray is traced. Refracted and reflected rays are shot from the point of intersection if the object is reflective or transparent. In addition, tracing of *shadow* ray towards a light source shows whether a given source lights a point or it is occluded. Finally, initial color of the material, results of reflected and refracted rays tracing, and lighting define pixel color.

Various ways of accelerating ray tracing have been invented for decades. In this paper we utilize some of the proposed ideas.

One of the most attractive features of ray tracing is its universality. Environments can consists of different objects such as polygons, splines, voxel grids, explicitly and implicitly defined surfaces etc. To make ray tracing on some sort of object possible, a procedure of ray - object intersection should be presented. Of course, this procedure ought to be as fast as possible, since the speed is often desired.

Abundant information on ray tracing can be found in [9], while this paper is dedicated to ray tracing on spatial patches.

1.2 What is Spatial Patch?

Spatial a patch is a new kind of rendering primitive. It can be defined as a dense range image of a small part of a surface. More formally, *spatial patch* (Figure 1) is defined by the origin point P, orthogonal frame $(\Delta x,\Delta y,\Delta z)$, and a rectangular $m\times n$ array of $(c,d)_{i,j}$ pairs, each representing a point $N_{i,j}=P+i\Delta x+j\Delta y+d_{i,j}\Delta z$ of color $c_{i,j}$ on a surface (some color value denotes transparency). We call this colored point a node. For each node $N_{i,j}$, a normal can be estimated as a normalized cross product of $N_{i+1,j} - N_{i-1,j}$ and $N_{i+1,j} - N_{i-1,j}$ vectors.

Surface can be regarded as a set of quadrangle cells $\{Q_{i,j}\}$, each having vertices in non-transparent points $N_{i,j}$, $N_{i+1,j}$, $N_{i+1,j+1}$, and $N_{i,j+1}$. All cells adjacent to a transparent node are regarded fully transparent.

Any 3D surface can be represented with the required level of accuracy by a set of spatial patches. We also assume, that each patch represents relatively small part of a surface.

Further details on the concept of spatial patches can be found in [1] or [2].

In the sequel, we make an additional assumption that $m = n = 2^k+1$, where k is non-negative integer. Given spatial patch of arbitrary m and n, it can be supplemented with transparent cells to meet this assumption.

Below, the initial coordinate system is referred as world (though, it is denoted as model in [1] and [2]). We will also operate with patch coordinate system, defined by patch orthogonal frame (see Figure 1).

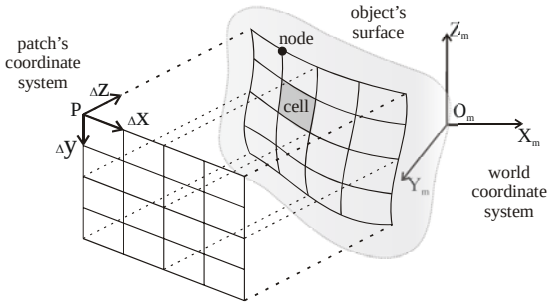

Figure 1: Spatial Patch definition

To extend ray tracing to the newly defined primitive, solution for typical ray - environment intersection problem should be found (i.e. an algorithm for finding intersection of an arbitrary ray with an arbitrary scene comprised of spatial patches should be presented).

1.3 Ray Tracing on Spatial Patches

As it was mentioned above, one can consider the whole scene as a set of quadrangle cells. By treating each cell as a pair of triangles or as a bilinear quadrangle, the scene can be reduced to either a

triangle set or to a set of explicitly defined surfaces – traditional environments for ray tracing. This straightforward algorithm has serious drawback: it has to operate with a vast number of tiny primitives, without exploiting their natural regularity.

In this paper we describe a technique of intersecting ray with a hierarchy (tree) of nested axes-aligned boxes, whereas this hierarchy has infinite depth and the intersection should be found with some pixel size-based precision. This approach, in our opinion, seems to be more natural and efficient for scenes represented by spatial patches. Construction of this hierarchy for 3D scenes is described in Section 2, whereas the process of ray-scene intersection is presented in Section 3. The paper concludes with discussion of our practical results.

2. SCENE TREE

In this section a scene represented by spatial patches (*a patched scene*) is considered. Basing on this scene an infinite-depth hierarchy of nested boxes called *scene tree* is created. The sequence of tree levels converges to the surfaces represented in a scene.

2.1 Upper Layer

To construct the upper layer of the tree we employ Goldsmith-Salmon [4] algorithm that allows construction of bounding volume hierarchies optimized for ray tracing. Restricting volume type to axes-aligned boxes, we obtain a tree of nested bounding volumes, in which each leave is a bounding box of a single patch. Hierarchies produced with Goldsmith-Salmon algorithm are proved to exploit scene coherences efficiently; therefore, they are often superior over conventional spatial subdivisions (octrees, regular grids), BSP-trees, etc. Nevertheless, these standard methods work with patched scenes efficiently, as well.

2.2 Patch tree

Obtained tree is finite and don't converge to a scene surface. Hence, each leaf, i.e. bounding box of a patch, should be replaced with an infinite subtree representing a surface defined by that patch. Let us call this subtree *a patch tree*. For this tree we use either world system axes-aligned boxes (*world-oriented* patch tree) or patch system axes-aligned boxes (*patch-oriented* patch tree) – see Figure 2. To each box in *a patch tree* we attribute a normal and a color, approximating mean color and normal throughout the surface enclosed in the box.

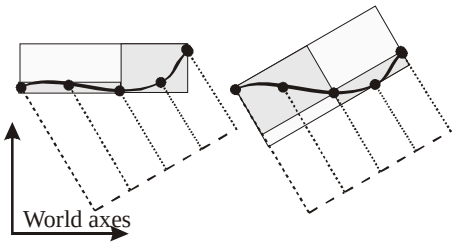

Figure 2: Patch layer for world-oriented (left) and patch-oriented (right) hierarchies, shown for 1D "patches" in 2D space.

2.2.1 Patch Layer

We recall, that we consider patches of $(2^k+1)\times(2^k+1)$ nodes in size. The upper box in the patch layer is simply a bounding box of all non-transparent nodes. After that, we subdivide our patch into four adjacent subpatches of $(2^{k-1}+1)\times(2^{k-1}+1)$ in size, and store their bounding boxes as children in a quadtree. Then this procedure is recursively repeated for the children until the size of 3×3 is reached.

Boxes, containing only transparent nodes, are expelled. For others, arithmetical mean of normals and colors for non-transparent nodes inside is looked up.

2.2.2 Cellular Layer

The lowest level in the patch layer is the level of 3×3 subpatches. In the next patch tree level bounding boxes for single cells are situated. To obtain lower levels of patch tree surface interpolation (subdivision), refining our patch, is required. In [1], [5], [7], and [8] some existing approaches to subdivisions and mesh interpolations are presented. Below we will distinguish nodes that are precalculated directly from patch (*primary* nodes), and nodes obtained during interpolation (*secondary* nodes). All calculations for nodes are performed in the same coordinate system as for boxes.

Below in the section we consider a node as a point in 9-dimentional space (3 dimensions for point, 3 -for normal, 3 -for color).

Consider an arbitrary cell. Denote its nodes by A, B, C, and D (Figure 3). Two types of recursive subdivisions, namely bilinear and smooth, may be used for interpolation. In both cases, on each step of recursive subdivision middle points of cell edges (nodes K, L, M, and N on Figure 3) and the central node (node O) are interpolated. For bilinear subdivision interpolation is straightforward:

$$K=\frac{A+B}{2}, L=\frac{B+C}{2}, M=\frac{C+D}{2}, N=\frac{D+A}{2},$$
$$O=\frac{A+B+C+D}{4}.$$

Thus, four smaller cells (interpolated cells of first generation) *AKON*, *KBLO*, *OLCM*, *NOMD* are obtained; their bounding boxes (calculated as bounding boxes of four points of their nodes) form the next level in the hierarchy. Each of these cells can be subdivided in the same way, if necessary.

The surface obtained during bilinear interpolation of cell interiors is not smooth on the cell edges, which, however, does not result in visible artifacts, provided that cells are of not more than several pixels size (the most often case). In the reminder cases, restoring a smooth surface over a whole patch is more suitable.

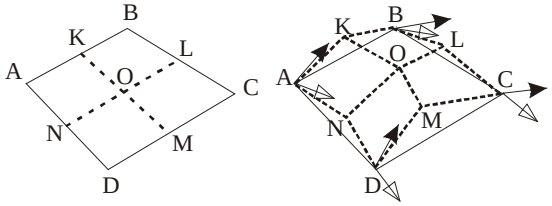

Figure 3: Recursive subdivision (left – bilinear, right – smooth)

In order to obtain such surface, we modified Kobbelt interpolatory scheme [8] developing a method of smooth interpolation of cell interior. In the case of smooth subdivision, we need to assign to each node two tangent 9-tuples: *dX* and *dY* – black and white arrows on the figure. For primary node $N_{i,j}$, $dX_{i,j} = (N_{i+1,j} - N_{i-1,j})/16$ and $dY_{i,j} = (N_{i+1,j} - N_{i-1,j})/16$, for secondary node they are interpolated.

Then, nodes are interpolated in the following way:

$$K = \frac{A+B}{2} + dX_A - dX_B, L = \frac{B+C}{2} + dY_B - dY_C,$$

$$M = \frac{C+D}{2} + dX_D - dX_C, N = \frac{D+A}{2} + dY_A - dY_D,$$

$$O = \frac{A+B+C+D}{4} + dX_A - dX_B$$

$$+ dY_B - dY_C + dY_B - dY_C + dY_B - dY_C.$$

Now smaller cells are obtained. For *A*, *B*, *C*, and *D* tangents are simply divided by two. For other nodes, tangent vectors are interpolated as follows:

$$dX_K = (B-A)/16, dY_K = (dY_A + dY_B)/2,$$

$$dX_L = (dX_B + dX_C)/2, dY_L = (C-B)/16,$$

$$dX_M = (C-D)/16, dY_M = (dY_D + dY_C)/2,$$

$$dX_N = (dX_A + dX_D)/2, dY_N = (D-A)/16,$$

$$dX_O = (M-K)/16, dY_O = (L-N)/16$$

In case of smooth interpolation, there arise the problem of finding a bounding box enclosing a surface within a cell. It can be solved for patch-oriented hierarchy by expanding a bounding box of cell nodes in z-direction by a value determined by tangent vectors. Bounding boxes in patched and upper layers should be expanded as well.

For each cell an average color and a normal are found as averages of cell nodes colors and normals.

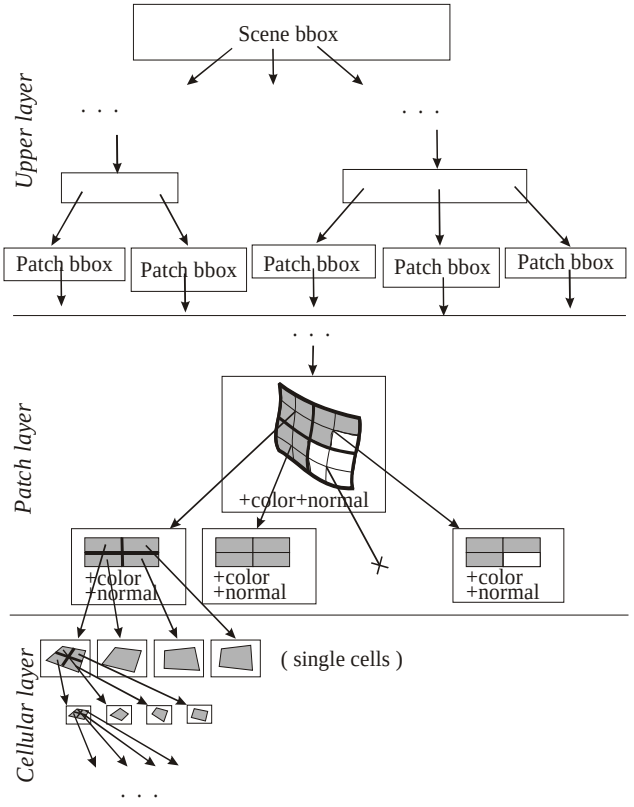

Figure 4: Scene Tree

In our implementation, all interpolated nodes are calculated on demand and are not stored. However, some cache techniques may be used. Furthermore, if before tracing some patch is known to be low refined (in the sense that its nodes are likely to require deep and frequent interpolations), all secondary nodes up to some generation level can be interpolated, stored, and later treated as primary ones. Still, this process is restricted by available amount of memory.

Thus, a scene tree is constructed (Figure 4). Logically, it has infinite depth, while, in practice, its lower levels in cellular layer are calculated up to a certain depth.

3. RAY – ENVIRONMENT INTERSECTION

3.1 Notations and Definitions

Assume a ray with a starting point *S*, and a direction vector *D* (|*D*| = 1). Let us attribute to it two additional values: an *initial threshold of details* T_0, and a non-negative *threshold of details increment* ΔT. To each non-negative *ray parameter t* we assign a point $P(t) = S+D*t$ and threshold of details $T(t) = T_0+\Delta T*t$. Threshold of details *T(t)* defines desired accuracy of operations (intersections) in the neighborhood of point *P(t)*. Thus, if $P(t) \in$ *screen_plane*, *T(t)* should equal $0.5 \times$ *size_of_pixel*.

An ordered pair *[a, b]*, where $0 \le a \le b$, define a line segment $\Delta_{ab} = [P(a), P(b)]$ on a ray. Such a segment is called a *ray segment*. The *segment threshold of details* denotes the value *T(a)*.

Let a ray segment Δ_{cd} be nested in a ray segment Δ_{ab}, if $a \le c \le d \le b$.

3.2 Ray Segment - Box Intersection

Consider an arbitrary axes-aligned box *B* and an arbitrary ray segment Δ_{ab}.

There are two possibilities: either they don't intersect or their intersection is a ray segment Δ_{cd} that is nested in Δ_{ab}. Well-known Cohen-Sutherland algorithm [3], extended with calculation of ray parameters of new points, is used here.

Assume a box and a segment intersect. If all dimensions of the box $x_{max} - x_{min}$, $y_{max} - y_{min}$, $z_{max} - z_{min}$ are less than segment threshold of Δ_{cd}, then the box *fits precision*. Otherwise, the box *doesn't fit precision*.

Thus, possible relative positions are classified into three categories: a box and a ray segment *don't intersect*, a box and a ray segment *intersect and the box fits precision*, a box and a ray segment *intersect and the box doesn't fit precision.*

3.3 Ray Segment – Scene Tree Intersection

Now, a problem of ray – environment intersection calculation is substituted for ray segment – scene tree approximate intersection problem calculation.

Intersection of a ray and a volume hierarchy is considered in a number of works ([4], [5], [6]). In our case the following recursive scheme is suitable:

```
IntersectionInfoStructure intersection;

bool Intersect( Ray Segment Δab,
                Scene Tree Node Node )
{
  Ray Segment Δcd = Δab ∩ Node.Box;
  if( Δcd is empty )
        return false;
  if( Node ∉ Upper Layer &&
      Node.Box.FitsPrecision(T(d)) )
  {
    intersection.Normal = Node.Normal;
    intersection.Color = Node.Color;
    intersection.Parameter = d;
    intersection.Point = P(d);
    return false;
  }

  bool result = false;
  for( every children Child )
  {
    if( Intersect(Δcd, Child) )
    {
       d = intersection.parameter;
       result = true;
    }
  }
  return result;
}
```

Thus, the process will stop when tree node' box fits the precision. Hence, *the less is desired precision the faster is intersection.*

Below peculiarities of ray segment – tree node intersection for boxes from different layers of the scene tree are observed.

3.4 Intersection with nodes of Patch Tree

There are a number of pre- and postprocessing operations required for this layer in some cases.

First, if with patch-oriented hierarchies are used, a ray should be converted to patch coordinates before intersecting with patch layer.

Besides that, on a postprocessing stage the intersection' normal vector needs to be normalized, as a normalized vector is desired for lightning and secondary rays shooting. In addition, in case of patch-oriented hierarchy, the normal vector as well as an intersection point should be converted back to world coordinates. Still, all these operations are computationally expensive. When consider an intersection of a ray segment with the whole scene, one should perform them only for the ultimate intersection, not for each intersected patch (there can exist several). In order to do that, a pointer to the nearest intersected patch should be stored along with other intersection parameters.

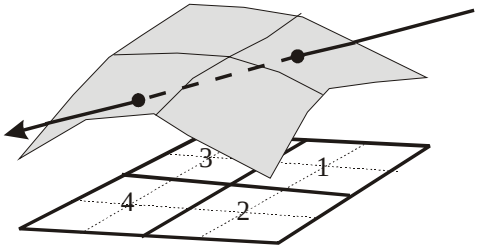

Figure 5: Optimized order of children traversal

One important optimization concerning the order of children traversal can be implemented as well. Analyzing the ray direction vector in patch coordinates, the optimal sequence of children traversal can be found. Indeed, a ray segment with a child possessing nearer box should be tested for intersection first and those with those of farther boxes should be tested last, since intersection with the latter is unnecessary if the former is intersected (Figure 5). Note that, such a sequence is constant over the whole patch tree.

3.5 Intersection with Upper Layer

Though a scene tree does not possess in the upper layer that regularity as in patch and cellular levels, optimizations in child traversal can be done. We implemented one of traditional techniques. On a preprocessing stage, we sort children boxes by their least coordinates of projections on six world semi-axes, thus obtaining six lists. When shooting a new ray, we find which of semi-axes lies closer to its direction vector, i.e. we find dominant direction of the ray. Now, intersection with the children, if performed in the sequence stored in respective list, may be stopped when the dominant coordinate of the next box exceeds that of current intersection.

3.6 Shooting ray segments

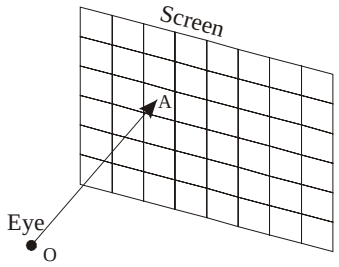

Figure 6: Primary ray

To generate a primary ray segment from the eye *O* to the pixel *A* (see Figure 6), a ray tracer ought to create a ray, having its starting point at *O*, its direction vector equal to $\overrightarrow{OA}/\left|\overrightarrow{OA}\right|$, its initial level of details equal to zero, and its level of details increment equal to maximum of pixel width and height divided by $2\cdot\left|\overrightarrow{OA}\right|$. The initial ray segment is $[\left|\overrightarrow{OA}\right|, \left|\overrightarrow{OA}\right| + 2*scene\_radius]$.

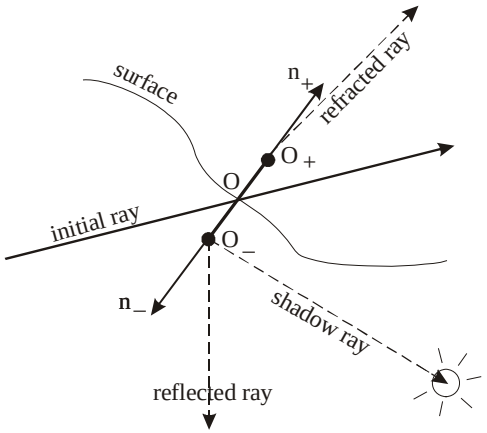

Figure 7: Secondary rays

To shoot a secondary ray, a normal at the point of intersection should be looked up. Assume that in the process of ray segment – scene tree intersection we get point *O* and a normal vector, which equals to either n_+ or n_- (in notation of Figure 7). To distinguish, is it n_+ or n_-, its dot product with the ray direction vector is computed. This value is positive for n_+ and negative for n_-. Thus,

we determine, which of two vectors we deal with, calculating the other.

Afterwards, reflected, refracted and shadow ray segments are generated. Direction vectors are calculated as usually in ray tracing. Initial thresholds of details are set to initial ray threshold of details at point O, while threshold of details increments remain the same. In order to avoid self-occlusions we can use O_+ as a starting point for refracted ray, and O_- for shadow and reflected rays. These points are obtained by a slight shift of point O along n_+ and n_- respectively by a threshold of details at point O (see Figure 7).

This scheme successfully solves self-occlusion problem. However, when a ray is almost exactly tangent to surface and the dot product of its vector with the normal vector is very close to zero, n_+ and n_- can be confused, thus causing an artifact pixel. Still, this effect is rare. Furthermore, it can be completely eliminated, if we state that a patch represents only one side of the surface (if necessary other side can be represented by another patch). In this case we always treat a normal vector obtained from intersection as n_-. If its dot product with the direction vector is positive then a patch is ignored (back face culling).

To conclude this section, we would like to emphasize that above a method of ray segment - patch tree intersection is given implicitly, and, thus, the ray – primitive intersection problem is solved. Therefore, a spatial patch becomes a full value ray tracing primitive. The following section proves its suitability for ray tracing.

4. PRACTICAL RESULTS

We have implemented a ray tracer based on the described above algorithm, which is capable of creating images of arbitrary patched scene. Since it was designed for testing and evaluation of tracing features peculiar to this kind of scenes, we did not supply it with all traditional acceleration techniques or techniques that enhance photo-realism. At the same time, experiments prove efficiency of our method in reducing of intersection to a sequence of segment - box clippings.

The scene presented at Figure 8 consists of two teapots each made of 2928 patches (patched model of the famous teapot was obtained from the polygonal one using the method described in [2]). Each patch has 17×17 node grid. Simple computations show that conversion to triangles would lead to a large (~3 million) number of them.

Figure 8: Ray tracing example

We traced our scene at 1024×768 and 256×192 resolutions. Nodes were interpolated bilinearly. The following table summarizes results for cases of patch-oriented and world-oriented patch trees.

The second column presents total numbers of traced rays that intersect at least upper box of the scene tree. The next column gives average number of Cohen-Sutherland ray segment – box clippings. The third column gives an average number of interpolated boxes (i.e. calculated tree nodes in cellular layer). The last column gives a time taken by tracing (preprocessing time not included). Calculations were performed on a Pentium II - 450 MHz processor.

Resolution and orientation	Number of rays	Clippings per ray	Interpolated boxes per ray	Tracing time (sec)
1024×768 (patch)	1,412,036	22.2	5.2	140
256×192 (patch)	87,755	17.3	0.06	6
1024×768 (world)	1,410,639	26.0	10.8	182
256×192 (world)	87,540	18.7	1.6	8

5. CONCLUSION

In this paper we presented a complete method of photo-realistic rendering of 3D scenes by means of ray tracing. In practice, it appeared to work efficiently due to key properties of spatial patches, such as small size and regular internal structure.

The use of hierarchy of nested boxes for spatial partitioning of patch clusters provides fast selection of the candidate patches that require further processing, i.e. calculation of the intersection points if intersection occurs. Goldsmith-Salmon approach for constructing of such hierarchy proved to yield a tree of boxes that allow for very fast processing. In addition, we expect that other known spatial partitioning strategies, including octrees, regular grids, BSP, and others, would work as efficient as the implemented technique.

The hierarchy of boxes within a patch seems to be quite natural for the proposed primitive. It utilizes patch's internal regularity and, hence, requires little additional storage space. Construction of this structure can be implemented efficiently, and, in fact, may be executed on-the-fly if memory resources are critical.

It is important that the proposed strategy allows for any kind of surface interpolation within a patch, including the least accurate piecewise linear, bilinear or more smooth ones. The resulting image can be obtained with any desired degree of accuracy; commonly known techniques of super- and stochastic sampling can be directly applied as well.

Thus, the representation of 3D models with spatial patches proved to be efficient not only for rendering using z-buffer (which is discussed in [1]), but also for photo-realistic rendering exploiting ray-tracing ideology. This fact significantly widens area of application of spatial patch technology, ensuring its compliance with various needs of 3D graphics practitioners.

6. ACKNOWLEDGMENTS

This work was carried on by Computer Graphics Group at the Mathematics Department of MSU under research agreement with Intel Technologies, Inc. We thank Jim Hurley and Alex Reshetov (Intel Technologies, Inc.) for their constant interest in this work.

7. REFERENCES

[1] D. Ivanov, Ye. Kuzmin. Spatial Patches – A Primitive for 3D Model Representation, *Proceedings of EuroGraphics*', 2001

[2] D. Ivanov, and Ye. Kuzmin. Representation of Real-life 3D Models by Spatial Patches. *In these proceedings*, 2001.

[3] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughs, "Cohen Sutherland Method", Computer Principles and Practice, *Addison Wesley*, Second Edition, 1990

[4] J. Goldsmith, J. Salmon, Automatic Creation of Object Hierarchies for Ray Tracing, *IEEE Computer Graphics and Applications*, 7:5, May 1987, pp. 14-20.

[5] S. Rubin, T. Whitted, A Three-Dimensional Representation for Fast Rendering of Complex Scenes, *ACM Computer Graphics (Proc. Of SIGGRAPH'80)*, 14(3), July 1980, pp. 110-116.

[6] T. Kay, T. Kajiya, Ray Tracing Complex Scenes, *ACM Computer Graphics (Proc. Of SIGGRAPH'86)*, 20(4), Aug 1986, pp. 269-278.

[7] E. Catmull, A Subdivision Algorithm for Computer Display of Curved Surfaces, Ph. D. Dissertation, *University of Utah*, Dec 1974.

[8] L. Kobbelt. Interpolatory Subdivision on Open Quadrilateral Nets with Arbitrary Topology. *Computer Graphics Forum (Proc. of Eurographics'96)*, 15, pp. 409-420.

[9] An Introduction to Ray Tracing, edited by Andrew S. Glassner, *Academic Press, New York*, 1989.

About the authors

Victor S. Lempitsky, Student – Vitya@fit.com.ru
Dr. Denis V. Ivanov, Scientist – Denis@fit.com.ru
Dr. Yevgeniy P. Kuzmin, Senior Scientist – Yevgeniy@fit.com.ru

Computational Methods Lab.
Mathematics and Mechanics Dept.
Moscow State University,
Vorobyovy Gory, Moscow, Russia, 119899