Convex object based volume visualization: a formal proof and example

Zou Qingsong\textsuperscript{a,}\textsuperscript{*}, Kwoh Chee Keong\textsuperscript{a}, Ng Wan Sing\textsuperscript{b}

\textsuperscript{a}School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, 639798, Singapore
\textsuperscript{b}School of Mechanical and Production Engineering, Nanyang Technological University, Nanyang Avenue, 639798, Singapore

Abstract

In this paper, we provide formal definitions for object space, space subdivision and ray tracing. We also introduce concept models such as ordered space subdivisions and ordered visualization. From these works, we can evaluate the correctness and performance of our space subdivision scheme for volume visualization. With these definitions, we also analyze some established space subdivision approaches such as uniform space subdivision and octree based space subdivision.

In this paper, we suggest a new subdivision strategy, the convex object based subdivision scheme, which can be more efficient in generating the convex objects for ordered visualization. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Volume visualization; Ray tracing; Space subdivision; Ordered space subdivision; Ordered visualization

1. Introduction

The traditional rendering method of visualizing a volumetric data set is to use surface based rendering. In this approach, a surface detector is applied to the volumetric data and fit geometric primitives of conventional surface rendering algorithms to obtain an intermediate representation. These techniques suffer from the common problems of having to make a binary classification decision when defining a surface. Another shortcoming is that only the surfaces are rendered without the internal data. To avoid these problems, researchers have begun exploring the notion of volume rendering. Volume rendering is capable of capturing the information on the outer surfaces as well as the details inside the volume. Levoy \cite{1} explored the application of volume rendering techniques to the display of surfaces from sampled scalar functions of three spatial dimensions. He also presented a front-to-back image-order volume rendering algorithm in \cite{2} where volume primitives could be directly projected onto a projection plane. Robb \cite{3} refined the original ray tracing algorithm for the implementation of a global illumination model, which involved the phenomena of reflections, refraction, shadows, and specular reflection. Since then, ray tracing has become one of the most powerful approaches to do volume rendering. However, this process is time and space consuming. Many techniques have been proposed to speed up its operation. Kohl \cite{4} proposed a technique to accelerate a ray tracing algorithm by applying a new technique for a fast computation of the intersections, called eye rays. Fujimoto \cite{5} introduced a general method of improving the computational speed of the ray tracing method to a level where the image synthesis time was practically constant by developing a traversing tool in the form of a line generator (3DDD). Upson \cite{6} used a method to continuously cover the gap between surface based rendering and volume rendering. Ke \cite{7} suggested a new traversal scheme to speed up ray tracing, which attempted to test bounding volumes from near to far.

There are techniques that explore the data structure to improve the rendering speed. A ray tracing implementation was described in \cite{8} that was based on

\textsuperscript{*}Corresponding author. Tel.: +65-8920617; fax: +65-7926559.
E-mail address: qszou@signet.com.sg (Z. Qingsong).

The main ideas behind these speedup techniques are to decrease the computation for rays and objects intersection tests. The use of ray and image coherence is the key point to speed up the ray tracing algorithms. The ray coherence is that when a ray hits a voxel or object, the rays that are nearer to the ray will more likely hit the same voxel or object also. The image coherence is that when a voxel or object is hit by a ray, the voxels or objects that are nearer to this one will be more likely hit by the same ray also. In particular, bounding volume and space subdivision are the two main schemes used in various speedup techniques. The bounding volume scheme tries to surround each object with a simple bounding volume. If a ray has no intersection with the bounding volume, it is not necessary to check for intersection with the object enclosed. Robb [3] used spheres as bounding volume; Crosnier [12] proposed to use a triboxx; Udupa [13] introduced a shell as the bounding volume. The space subdivision scheme subdivides the scene into small cells pointing to lists of objects. The intersection between a ray and the objects, which lie near the ray path, suggests that the objects are more likely to be intersected by the ray. There are two kinds of space subdivision techniques, uniform space subdivision shown by Cleary [9] and nonuniform space subdivision shown in [8,11–13,17]. The nonuniform space subdivision strategy allows more efficient adaptive subdivision of space by taking care of the spatial coherence of the objects space while uniform space subdivision will show advantages in locating the next node along the ray tracing path because of the uniform size of the subdivided cells.

From the reviewing of all speedup techniques, we find that the formal definition of volume rendering, which is critical for formal method processing, was not provided. None of the researches has proposed formal definitions for object space, space subdivision and ray tracing. So it is difficult to evaluate the efficiency of a subdivision scheme performance and its correctness.

In this paper, we will provide formal definitions of the above-mentioned objects and some other new ideas and use these ideas to analyze some space subdivision approaches such as pyramid ray tracing, cell based space subdivision and octree based space subdivision schemes. Besides, we also suggest a more efficient space subdivision strategy.

In Section 2, we will give the formal definition of object space and scene space. In Section 3, the definition of space subdivision will be given, new ideas of Hull of space subdivision and ordered subdivision will be introduced. In Section 4, we will introduce the ideas of ray space and a special ray space parallel ray. In Section 5, we will introduce the definition of volume visualization and the idea of ordered visualization. Convex space subdivision based visualization will be discussed in this section also. In Section 6, we will propose a new convex object generation algorithm. In Section 7, we will give the results and discussion of our research followed by the concluding remarks in Section 8.

2. Object space and scene space

In this section, we introduce our formal definitions related to volume visualization and space subdivision schemes. The first definition is of object space.

2.1. Object space

Object space is the whole space that contains all objects and rays to be rendered in the scene. In our discussion, due to the characteristics of computer graphics, the object space is a finite discrete space.

Object space denoted as $\Theta$ is defined as follows:

- $\Theta$ is the object space.
- $\Theta$ is a finite set.
- $|\Theta|$ is the size of set $\Theta$.

$N$ is the set of natural numbers and $\Theta$ can be defined as

- $\Theta = \{o_1, o_2, \ldots, o_i, \ldots, o_n\}, i \in N, n = |\Theta|$, $\forall o_i \in \Theta$, which is the minimum primitive of object space, we denote it as voxel,
- $\forall o_i \in \Theta$, we associate $P$ with it as a special property, such as the density or color of a voxel.

In the later section, we will use $P$ to identify an important feature of space subdivision.

Since the object space $\Theta$ is a finite discrete space, there exists an order $R_{os}$, which can be used to enumerate the elements of $\Theta$. The order $R_{os}$ is defined as follows:

- $\exists R_{os}$, $R_{os}$ is a linear order over the voxels of object space $\Theta$.
- $\forall o_i \in \Theta, o_i \neq o_j \Rightarrow o_i R_{os} o_j \text{ or } o_j R_{os} o_i$. 

In the later section, we will use $P$ to identify an important feature of space subdivision.
We call $R_{os}$ as space order and we can use $R_{os}$ to enumerate all elements of object space one by one.

Some examples of scene space are outlined here. Fig. 1 shows a 2D Cartesian coordinates system. Hence, for Fig. 1,

- $\Theta = \{<x_p, y_k>| p \in [0, 9], k \in [0, 4]\}$.
- $|\Theta| = 50$.

For convenience, we denote $<x_p, y_k>$ as $o_{p,k}$.

How to define $R_{os}$ for the 2D Cartesian coordinates system? Before giving the definition of $R_{os}$, we give the definition of natural axis order (NAO).

NAO is defined as

- $\forall x_p \in x, \forall x_q \in x, x_p \neq x_q$,
- $p < q \Rightarrow x_p \text{NAO}_x x_q$,
- $q < p \Rightarrow x_q \text{NAO}_x x_p$.

Now, we can define $R_{os}$ of 2D Cartesian coordinates system as follows:

$$R_{os} = \langle \text{NAO}_x, \text{NAO}_y \rangle$$

Fig. 2 shows an example object space of a 3D Cartesian coordinates system. Hence, for Fig. 2,

- $\Theta = \{<x_p, y_k, z_t>| p \in [0, 9], k \in [0, 4], t \in [0, 5]\}$,
- $|\Theta| = 300$,
- $R_{os} = \langle \text{NAO}_x, \text{NAO}_y, \text{NAO}_z \rangle$.

For convenience, we denote $<x_p, y_k, z_t>$ as $o_{p,k,t}$.

A 2D polar coordinates system is outlined in Fig. 3. For Fig. 3, we have

- $\Theta = \{<r_p, \theta_k>| p \in [0, 3], k \in [0, 15]\}$,
- $|\Theta| = 64$,
- $R_{os} = \langle \text{NAO}_r, \text{NAO}_\theta \rangle$.

A 3D polar coordinates system is outlined in Fig. 4. For Fig. 4,

- $\Theta = \{<r_p, \theta_k, \tau_t>| p \in [0, 3], k \in [0, 15], t \in [0, 15]\}$,
- $|\Theta| = 1024$,
- $R_{os} = \langle \text{NAO}_r, \text{NAO}_\theta, \text{NAO}_\tau \rangle$.

In this paper, we will take the 2D cartesian coordinates system as the default object space, denoted as 2DCCS; we denote 3D Cartesian coordinates system as 3DCCS; 2D polar coordinates system as 2DPCS and 3D polar coordinates system as 3DPCS, respectively.

2.2. Scene space

In the preceding section, we have discussed object space, which is the whole space of objects and rays. As we know, volume visualization is a procedure of using rays to intersect with the objects. There are two kinds of entities related to volume visualization, objects and rays. In this section, we will first discuss the entity known as objects. We define a new entity—scene space. Scene space is a subset of the object space, which can be considered as a special instance of object space. In our research, there is only one object space but always more than one scene space. Scene space is also a finite discrete set. The definition of scene space of 2DCCS is shown as follows:

- $S$ is a scene space.
- $S \subseteq \Theta$.

Fig. 5 gives an example of a scene space of object space outlined as Fig. 1. Hence for Fig. 5,

- $S = \{o_{0,1}, o_{0,2}, o_{0,3}, ..., o_{8,2}\}$,
- $|S| = 21$.  

Fig. 1. An object space of 2D Cartesian coordinates system.

Fig. 2. An object space of 3D Cartesian coordinates system.

Fig. 3. An object space of 2D polar coordinates system.

Fig. 4. An object space of 3D polar coordinates system.

Fig. 5. An example of a scene space of object space outlined as Fig. 1. Hence for Fig. 5.
3. Space subdivision

As brute-force volume rendering is slow, we would like to use some space subdivision scheme to speed it up. In this section, we will give formal definitions related to space subdivision. There are two kinds of space subdivisions; object space subdivision and ray space subdivision. Most of the implementations use object space subdivision scheme and so in our research, we will focus on object space subdivision.

3.1. Space subdivision

Let us assume that we have an object space \( Y \) and correspondingly, we have a scene space \( S \). We define \( V \) as a subdivision scheme over scene space \( S \); which is defined as follows:

- \( V = \{v_1, v_2, \ldots, v_n\} \) is a subdivision of \( S \).
- \( V \) is a finite set.
- \( \forall v_i \in V \Rightarrow v_i \subseteq S \).
- \( \forall v_i \in V, \forall v_j \in V, v_i \neq v_j \Rightarrow v_i \cap v_j = \phi \).
- \( S = \bigcup_{i=1}^{\mid V \mid} v_i \).

For convenience, in a later part, we call \( v_i \) as a subdivision cell.

A general subdivision scheme is outlined in Fig. 6, in which some subdivision cells will have more than one voxel; for example, \( v_1 \) has five voxels. We call this kind of scheme object based subdivision. Hence, for Fig. 6,

\[ V = \{v_1, v_2, \ldots, v_9\}, \]
\[ v_1 = \{o_{0,1}, o_{0,2}, o_{0,3}, o_{1,2}, o_{1,3}\}, \]
\[ v_2 = \{o_{2,2}, o_{2,3}, o_{3,1}, o_{3,2}, o_{3,3}, o_{4,2}, o_{4,3}, o_{5,1}\}, \]
\[ v_3 = \{o_{5,2}, o_{5,3}, o_{6,2}, o_{6,3}, o_{7,1}, o_{7,2}, o_{7,3}, o_{8,2}\}, \]
\[ |V| = 3. \]

A special subdivision scheme is outlined in Fig. 7. This subdivision scheme treats every voxel as a subdivision cell. It is a special case of object based subdivision scheme. We call it voxel based subdivision. Hence, for Fig. 7, we have

\[ V = \{v_1, v_2, \ldots, v_{21}\}, \]
\[ v_1 = \{o_{0,1}, o_{0,2}, \ldots, v_{21} = o_{8,2}\}, \]
\[ |V| = 21. \]

For convenience, we denote \( v_i = \{o_{p,k}\} \) as \( v_{p,k} \).

3.2. Hull of space subdivision

In Section 2, we understand that every space subdivision scheme will subdivide the object space into many small subdivision cells. The problem now is, how to identity different cells? So, we introduce the concept of Hull.

As mentioned in Section 2, \( P_i \) is the associated property of voxel \( o_i \). We define \( P_{\text{min}} \) and \( P_{\text{max}} \) as the minimum and maximum thresholds, respectively. \( M_i \) is the neighborhood set of \( o_i \) for \( \forall o_i \in V, \forall v_j \in V \) and so \( |M_i| \) is the voxel number of neighborhood set. The definition of Hull is as follows:
• \( \forall v_i \in V, \forall o_i \in v, |M_i| \times P_{\min} \leq \sum_{o \in M_i} P_o \)
  \( P_o < |M_i| \times P_{\max} \Rightarrow o_i \in H_r \).
• \(|H| = |V|\).

Hull of 2DCCS is defined as follows:

• \( \forall v_i \in V, \forall o_{ij} \in v, 4 \times P_{\min} \leq \sum_{p \in [0,1]} q \in [0,1] \)
  \( P_{ij} \leq 4 \times P_{\max} \Rightarrow o_{ij} \in H_r \).

Every subdivision cell has a corresponding Hull, which includes all primitives that are located along the outline. We can identify each subdivision cell by its Hull and the Hull can be taken as the surface of a subdivision cell.

3.3. Convex space subdivision and concave space subdivision

The Hulls of 2DCCS are polygons. If all Hulls of 2DCCS are convex polygons, we call it a convex space subdivision or convex object based subdivision; otherwise, we call it concave space subdivision. As to 3DCCS, the Hull will be a polyhedron. Fig. 8 gives an example of convex space subdivision and concave space subdivision.

3.4. The order of space subdivision

Object space is the whole space consisting of many scene spaces and we can use an order \( R_o \) to enumerate the element of the object space. The scene space can be divided into many subdivided cells according to some subdividing scheme and we can use the Hull to describe these cells. Now, we want to define an order for the subdivision cells so that we can manage subdivision cells for efficient rendering. This section will focus on this issue. We denote this order as \( R_v \), which is defined as follows:

• \( \exists R_v, R_v \) is a linear order over the subdivision cells of subdivision \( V \) of scene space \( S \).
• \( \forall v_i \in V, \forall v_j \in V, v_i \neq v_j \Rightarrow \) either \( v_i R_v v_j \) or \( v_j R_v v_i \).

We call \( V \) an ordered subdivision of scene space \( S \) with the order \( R_v \). We can extend the definition of \( R_v \) to the voxel of every subdivision cell as follows:

• \( \forall v_i \in V, \forall v_j \in V, v_i \neq v_j, \forall o_p \in v_i, \forall o_q \in v_j, v_i R_v v_j \Rightarrow o_p R_v o_q. \)

\( D_v = \langle V, R_v \rangle \) is a linearly ordered set or chain, we call it subdivision chain.

The ordered set of subdivision is denoted as \( D_v \) defined as follows:

• \( D_v = \{ V_1, V_2, \ldots, V_i, \ldots, V_n \} \).
• \( \forall V_i \in D_v, \forall V_j \in D_v, i < j \Rightarrow V_i R_v V_j. \)

\( D_v^* \) is the set of all lists of finite length with symbols from \( D_v \).

• \(|D_v^*| = 2^{|D_v|}\).

In the object space subdivision in Fig. 6, we can define \( R_o \) as \( R_o = v_1 R_v v_2, v_2 R_v v_3 \) and hence, we have

• \( D_v = \{ V_1, V_2, V_3 \} = \{ v_1, v_2, v_3 \} \).
• \( D_v^* = \{ \phi, v_1, v_2, v_3, v_1 v_2, v_1 v_3, v_2 v_3, v_1 v_2 v_3 \} \).

In the voxel based subdivision outlined in Fig. 7, we define \( R_v \) as \( R_v = R_o \).

We define a new order \( R_v = \langle R_v, R_v^* \rangle \); based on order \( R_v \), we can enumerate all the subdivision cells and all the voxels in a subdivision scheme. Does there always exist an order for any space subdivision scheme in convex space subdivision and concave space subdivision? We will address this issue in later sections.

Fig. 7. Voxel based subdivision of Fig. 5.

Fig. 8. Convex space subdivision and concave space subdivision.
4. Ray space and ray

Volume visualization procedures require us to find the intersection between rays and objects. In this section, we will introduce a new entity, called ray space, which includes all the rays in a visualization process.

We define a ray RS as follows:

- RS is a ray space.
- RS \( \subseteq \emptyset \).

For every ray space RS, there exists a partition \( R \) defined as follows:

- \( R = \{ r_1, r_2, \ldots, r_t, \ldots, r_n \} \).
- \( R \) is a finite set.
- \( \forall r_j \in R, \forall r_j \in R, r_j \neq r_j \Rightarrow r_j \cap r_j = \emptyset \).
- \( RS = \bigcup_{j=1}^{n} r_j \).
- \( \forall r_j \in R \Rightarrow r_j \in RS \).

For convenience, we can denote \( \forall r_j \in R \) as a ray.

Now we want to introduce the idea of parallel rays, a special case of ray space. Parallel rays over a 2DCCS is defined as follows:

- \( \forall r_j \in R, \forall r_j \in R, r_j \neq r_j \Rightarrow r_j R o r_j \Rightarrow r_j \neq r_j \Rightarrow p \neq r_k \).
- \( \forall r_j \in R, \forall o_{p_k} \in r_j, \forall o_{r_i} \in r_j \Rightarrow p \neq r, k \neq 1 \).

Fig. 9 is an example of parallel ray of 2DCCS.

Similar to scene space, there is an order of the partition of ray space and we denote it as \( R_o \), which is defined as follows:

- \( \exists R_o, R_o \) is a linear order over the elements of rays of ray space,
- \( \forall r_j \in R, \forall r_j \in R, r_j \neq r_j \Rightarrow (r_i, R_o, r_j) \) or \( (r_j, R_o, r_i) \).

For \( \forall r_j \in R, \forall r_j \in R, r_j \neq r_j \Rightarrow o_{p_k} \in r_j, \forall o_{r_i} \in r_j \).

As to parallel rays, according to definition, we have \( p \neq r \) and so we define \( R_o \) as follows:

\[ r_i R_o r_j, \quad p < r, \]
\[ r_j R_o r_i, \quad p > r. \]

We extend the definition of \( R_o \) to the elements of every ray as follows:

\[ \forall r_j \in R, \forall r_j \in R, r_j \neq r_j, r_i R_o r_j, \forall o_{p_k} \in r_j, \forall o_{r_i} \in r_j \Rightarrow a_k R_o a_j. \]

Now, we define a new order \( R_v = \langle R_o, R_{v_o} \rangle \) so that we can use order \( R_v \) to enumerate the elements of every ray and every ray element.

5. Volume visualization and ordered visualization

Up to now, we have introduced all entities such as scene space, space subdivision and ray space. In this section, we will focus on the procedure of finding the intersection of scene space and object space.

5.1. Volume visualization

Volume visualization is using rays cast from the viewpoint to illuminate the scene space. It can also be understood as finding the intersection between voxels of scene space and rays of ray space. We define volume visualization as follows:

- \( VV_1 = S \cap R \) (1)
- or \( VV_2 = R \cap S. \) (2)

The intersection is ordered which means \( VV_1 \neq VV_2 \) because we have used some object space subdivision scheme to subdivide the scene space to smaller subdivision cells and we also use the ray partition scheme to subdivide the ray space. As to (1), we use every ray to illuminate the scene space; this is called ray based visualization. As to (2), we enumerate the scene space first and use every object of scene space to intersect with the ray space, which is called object based visualization.

As to ray space, from previous discussion, we know we can use order \( R_v \) to enumerate all the voxels of ray space, which means we can firstly use the order \( R_o \) to enumerate the rays and then use the order \( R_{v_o} \) to enumerate all the voxels of every ray. As to scene space, if we have an ordered subdivision scheme, we can use the order \( R_o \) to enumerate all the voxels, the order \( R_v \) to enumerate all subdivision cells and then use \( R_{v_o} \) to enumerate all voxels.

Usually, we will use ray based visualization scheme for subdivision schemes such as octree, uniform space subdivision, etc. In our research, we use object based visualization instead. What is the advantage of object space visualization over ray based subdivision? The idea comes from the fact that in most volume representation, most of the object space is empty and most of the time, the user just wants to see some specific part instead of the whole space. Algorithms using ray based visualization will use every ray to intersect the object space, so it cannot make good use of the characteristic of the object space. On the other hand, object based visualization will use the object to intersect the ray; we can only use the objects we are interested in to do the visualization and so
get a speedier visualization rendered result. From the discussion, we can understand that object based visualization is actually a reverse version of ray based visualization. So if we take every ray of ray space as a counterpart of space subdivision cell of scene space, we can consider ray based visualization as object based visualization and so we can use the same method to evaluate ray based visualization and object based visualization. Ray based visualization uses the coherence of rays to do visualization while object based visualization will use image coherence to do visualization. A more detailed comparison and some example will be given in the discussion section.

Object based visualization is defined as follows:

- \( VV = \bigcup_{j=1}^{V} t_j, t_j = v_j \cap R = \bigcup_{p=1}^{R} (v_j \cap r_p). \)

Hence,

- \( VV = \bigcup_{j=1}^{V} \bigcup_{p=1}^{R} (v_j \cap r_p). \)
- \( |VV| = |V|. \)
- \( VV \subseteq \Theta. \)

Fig. 10 shows an example using parallel ray \( R \) of Fig. 9 to visualize scene space \( S \). By Fig. 5, Hence, for Fig. 10,

- \( VV = \{i_1, i_2, i_3\}. \)
- \( i_1 = \{o_{0,1}, o_{0,2}, o_{0,3}, o_{1,2}, o_{1,3}\}. \)
- \( i_2 = \{o_{2,2}, o_{2,3}, o_{3,1}, o_{3,2}, o_{3,3}, o_{4,2}, o_{4,3}, o_{5,1}\}. \)
- \( i_3 = \{o_{5,2}, o_{5,3}, o_{6,2}, o_{6,3}, o_{7,1}, o_{7,2}, o_{7,3}, o_{7,2}\}. \)

5.2. Ordered visualization

In this section, we will discuss the order of visualization. If \( V \) is an ordered subdivision over the Scene \( S \) with the order \( R \), we can generate a rendering chain \( I \) for every ray. The following is the generated rendering chain for Fig. 10, the rendering chain being an ordered chain, which means

- \( \forall I_i \in I, \forall v_j \in \Gamma, \forall v_j \in I_i, v_j \neq v_j, v_j \) precedes \( v_j \Leftrightarrow \) ray \( r_i \) hits subdivision cell \( v_j \) before it hits subdivision cell \( v_j \).

Hence, for Fig. 10, we have

- \( I = \{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9\}. \)
- \( I_1 = I_2 = \{v_1\}. \)
- \( I_3 = I_4 = I_5 = \{v_3\}. \)
- \( I_6 = \{v_2v_3\}. \)
- \( I_7 = I_8 = I_9 = \{v_3\}. \)

From the above, we know that rays \( r_1 \) and \( r_2 \) hit only one subdivision cell \( v_1 \), rays \( r_3, r_4, r_5 \) hit only one subdivision cell \( v_3 \), ray \( r_6 \) hits two subdivision cells, it hits \( v_3 \) before it hits \( v_5 \), rays \( r_7, r_8, r_9 \) hit only one subdivision cell \( v_3 \).

If \( I \in D^*_r \), then the subdivision \( V \) over the scene space \( S \) with the order \( R \), visualized by the ray space \( R \) is an ordered visualization. Ordered visualization means we can illuminate the scene space one subdivision cell by another subdivision cell.

The example outlined in Fig. 10 is an ordered example.

If we define \( R \) as \( v_1R_1v_3, v_3R_2v_2 \), hence,

- \( D_0 = \{V_1, V_2, V_3\} = \{v_1, v_3, v_2\}. \)
- \( D^*_0 = \{\phi, v_1, v_2, v_3, v_1v_2, v_1v_3, v_3v_2, v_1v_3v_2\}. \)

However, since \( I_0 \notin D^*_r \), \( I \notin D^*_r \), so it is an unordered visualization. Note that only when the visualization is ordered, we can visualize the scene space one subdivision cell after another; on the other hand, if it is not an ordered visualization, we cannot get the correct visualization result.

From the above discussion, we know that in order to perform ordered visualization, we must first define the order \( R_0 \), which is the basic order for scene space and ray space. Then we define the appropriate order \( R \) of ray.

The next task in this derivation is to perform subdivision to obtain the order \( R \) of subdivision cells. In our research, the object space is 2DCCS and ray space is parallel ray. The last objective is to find a subdivision scheme that provides the appropriate order \( R_\alpha \).

First, we will prove that convex space subdivision generates appropriate order for ordered visualization.

5.3. Convex space subdivision based visualization

In the section on defining the order of space subdivision, we raised a question, “Does there always exist an order for any space subdivision scheme?” The answer is “no”. In this section, based on the above discussions, we want to prove that convex object based space subdivision is sufficient for an ordered volume visualization, which is illustrated in the following with parallel rays used to visualize the objects.

**Theorem 1.** \( V \) is a convex subdivision of scene space \( S \), \( PR \) is the parallel ray, \( r_0 \in PR \), \( VV \) is the visualization with rendering chain \( I = \langle PR, S \rangle \), \( \exists v_1, v_2, v_1 \neq v_2, v_1 \in I \times (r_0 \cap s), v_2 \in I(r_0 \cap s), v_1R_0v_2 \) for \( r_0, \forall r, r \in R, v_1, v_2 \in I \times (r \cap s) \Rightarrow \forall R, v_2 \) for \( r \).
In 2DCCS, the above theorem can be explained as $V$ is a convex subdivision of scene space $S$. PR is the parallel ray used to visualize the objects. $r_0 \in PR, I = \langle PR, S \rangle$, $\exists v_1, v_2, v_1 \neq v_2, v_1$ and $v_2$ intersect $r_0$, $v_1$ intersects $r_0$ before $v_2$ intersects $r_0$. Then for every $r$ of PR, if it intersects both $v_1$ and $v_2$, it must intersect $v_1$ before $v_2$.

We can prove it as follows:

If Theorem 1 is false then there exists one $r$, which intersects both $v_1$ and $v_1$, but it intersects $v_2$ before it intersects $v_1$. If $r$ intersects $v_1$, then there exist 1 or 2 intersection points except $r$ overlay an edge of $v_1$. If we take $o_{1,1}$, one of the intersection points for an example, $o_{2,1}$ for $v_2 \cap r$, $o_{1,2}$ for $v_1 \cap r$, which is outlined in Fig. 11, $o_{1,1}, o_{2,2}, o_{1,2}$ and $o_{2,1}$ will construct a rectangle, as anti-clockwise, if we connect $o_{2,1}$ and $o_{2,2}$, we get a line $o_{2,1}o_{2,2}$. Since $o_{2,1}, o_{2,2}\in v_2$ we can get line $o_{1,1}o_{1,2}$ and $o_{1,1}o_{1,2}\in v_1$. As $o_{1,1}o_{1,2}$ and $o_{2,1}o_{2,2}$ are sure to intersect, we call the intersection point $o_1$, and so we get $\bullet o_{1}\in o_{1,1}o_{1,2} \Rightarrow o_{1}\in v_1$ and $o_{1}\in o_{2,1}o_{2,2} \Rightarrow o_{1}\in v_2$.

This contradicts with $v_1 \cap v_2 = \emptyset$.

From Theorem 1, we know that in convex subdivision, if one ray intersects one subdivision cell before all other subdivision cells, all other rays must intersect that cell before others too. However, this is not true for concave subdivision. Fig. 12 illustrates the conjecture. $v_1$ and $v_2$ are two subdivision cells; $r_1$ and $r_2$ are two rays and both of them intersect $v_1$ and $v_2$. But we can see that $r_1$ intersects $v_1$ before it intersects $v_2$. On the other hand, $r_2$ intersects $v_2$ before it intersects $v_1$.

Hence, only convex space subdivision can cause ordered visualization and there must exist an order $R$ for convex space subdivision so that all the rays can visualize the subdivision cells one by one.

The following section will introduce the method to determine the order $R$.

5.4. Subdivision chain generation of convex space subdivision

To determine the order $R$, we can use the method we have introduced to enumerate the subdivision cells and extend that to all the objects to get an ordered chain. This procedure is called subdivision chain generation. In order to generate the subdivision chain of convex space subdivision, we must define the order $R$. The following is the technique to define $R$. For each subdivision cell $v$, we can find two corner voxels, denoted as $x_{\min}$, $x_{\max}$, such that

- $\forall o_p \in v \Rightarrow x_{\min} < x_p < x_{\max}$
- $\forall o_p \in ST \Rightarrow x_{\min} < x_p < x_{\max}$.

Extending $x_{\min}$ and $x_{\max}$ parallel to the $y$-axis, we will get two lines and a strip between these two lines, denoted as ST, outlined in Fig. 13.

In the two subdivision cells $v_1, v_2$, if $ST_1 \cap ST_2 = \emptyset$, we define $ST_{1,2} = ST_1 \cap ST_2$, and $y_{\min1,2}$ and $y_{\max1,2}$ so that

- $\forall o_p \in ST_{1,2}, o_p \in v_1 \Rightarrow y_{\min1,2} < y_p$, $\forall o_p \in ST_{1,2}, o_p \in v_2 \Rightarrow y_{\min1,2} < y_p$.

as outlined in Fig. 14.

Now, the entity $R$ is defined as follows:

- $\forall ST_1 \cap ST_2 = \emptyset$, we say that $v_1$ and $v_2$ are free of order, denoted as $v_1$ FOO $v_2$, which means we can assign either $v_1 R v_2$ or $v_2 R v_1$.

If $ST_1 \cap ST_2 \neq \emptyset$ from Theorem 1, we know Eq. (1) $y_{\min1,2} < y_{\min2,1}$, $\forall o_p \in ST_{1,2}, o_p \in v_1 \Rightarrow o_p \in y_{\min1,2}$, and so we can give the definition of $R$, as follows:

- $y_{\min1,2} < y_{\min2,1} \Rightarrow v_1 R v_2$
- $y_{\min1,2} < y_{\min2,1} \Rightarrow v_2 R v_1$

The definition of $R$ is illustrated in Fig. 15. Now, we have the definition of $R$, of convex space subdivision. Next, we will discuss how to generate the subdivision chain from that order. The chain generation is a reverse procedure of space subdivision:

- Find all the possible ordered cell couples according to the definition of $R$.

Fig. 12. Unordered visualization of concave subdivision.

Fig. 13. One subdivision cell of convex space subdivision.
• Combine all the possible cell couples into subchain according to the order;
• Recursively combine all the possible subchains until the subdivision chain is formed. If this process cannot be continued because the last subchains are FOO, we can combine the subchains according to the prefix.

The following is an example of subdivision chain generated using the above technique, as outlined in Fig. 16.

From the definition of $R_i$, we get the following cell couples:

- $v_2 R_i v_1, v_1 R_i v_3, v_2 R_i v_3, v_2 R_i v_4, v_4 R_i v_5, v_7 R_i v_6$.

We generate the subchains as follows according to the method introduced:

- $v_2 v_1 v_3, v_2 v_4 v_5, v_7 v_6$.

The last subchain cannot be combined together because they are FOO and so, we need to combine the subchains according to the prefix. Since subchains $v_2 v_1 v_3, v_2 v_4 v_5$ have the same prefix $v_2$, we can combine them into either $v_2 v_1 v_3 v_4 v_5$ or $v_2 v_4 v_5 v_1 v_3$ arbitrarily. Here we will take the first one. Finally, we combine the last subchains $v_2 v_1 v_3 v_4 v_5$ and $v_7 v_6$ to generated the subdivision chain:

- $D_i = \{v_2, v_1, v_3, v_4, v_5, v_7, v_6\}$.

Once the subdivision chain is generated, it is the order for visualization order of subdivision cells. Every time, we can pick a subdivision cell from the chain head and use all rays to render it.

In our research, we want to perform adaptive visualization. This means we subdivide the scene space into some small parts, the subdivision cells. When we need more detailed information of one subdivision cell, we need to subdivide it further. This process is called recursive subdivision of space subdivision. The next part will prove that we can subdivide a subdivision cell further and insert the new generated chain into the whole subdivision chain without losing the order.

5.5. Recursive subdivision of convex space subdivision

If $V$ is a convex space subdivision, we can subdivide every subdivision cell further using convex space subdivision scheme without losing the order. This can be illustrated formally (Fig. 17) as follows:

Suppose $V = \{v_1, v_2, \ldots, v_i, \ldots, v_n\}$ is a convex subdivision.

**Theorem 2.** $D_c = \{V_1, V_2, \ldots, V_i, \ldots, V_n\}$ is the subdivision scene, from the definition of subdivision chain, we have $\forall v_i \in D_c \Rightarrow V_i R_i V_{i+1}$. If we subdivide a subdivision cell (we will take $V_i$ as an example without losing generality), according to the convex space subdivision scheme (we assume $V_{\phi} = \{V_{i,1}, V_{i,2}, \ldots, V_{i,p}, \ldots V_{i,\phi}\}$ is such a subdivision scheme), the resulted order must be true for $\forall V_{i,p} \in V_{\phi} \Rightarrow V_{i-1} R_i V_{i,p}$ and $V_{i,p} R_i V_{i+1}$.

First, we will prove $V_{i-1} R_i V_{i,p}$.

If $ST_{i-1} \cap ST_{i,p} = \phi$ then $V_i$ FOO $V_{i,p}$; the assertion is true;

If $ST_{i-1} \cap ST_{i,p} \neq \phi$, because $V$ and $V_i$ are convex space subdivision, we know

- $ST_{i,p} \subset ST_i$ and $y_{\min i,p,i-1} \in ST_{i,p} \Rightarrow y_{\min i,p,i-1} > y_{\min i,i-1}$.
- $ST_{i-1,i,p} = ST_{i-1} \cap ST_{i,p} \Rightarrow ST_{i-1,i,p} \subset ST_{i-1} \cap ST_i \Rightarrow ST_{i-1,i,p} \subset ST_{i-1,i}$.

Now, we have

Fig. 14. Two subdivision cells of convex space subdivision.

Fig. 15. Definition of $R_i$ of convex space subdivision.

Fig. 16. An example of subdivision chain generation of convex space subdivision.
Also, we know \( y_{\min i,i-1,p} > y_{\min i,i-1} \).
And so, \( y_{\min i-1,p} < y_{\min i,i-1,p} \Rightarrow V_i^{-1} R_i V_i \).
Similarly, we can also prove \( V_{i,p} R_{i,p} V_{i+1} \).

From Theorem 2, we know we can divide the subdivision procedure into several stages. At each stage, we can subdivide a subdivision cell into smaller subdivision cells and insert the chain into the whole subdivision chain. We can use this technique to perform adaptive subdivision or to analyze the proposed subdivision technique for its correctness.

6. Convex object generation

Usually, the scene space consists of different kinds of objects. In general, most of them are concave objects. For example, in our research, we will allow the users to subdivide the object space and define the objects as they like to generate a list of objects that have meanings to them. After the user defined the objects, we found that, most of them are concave objects. From the discussion above, we know convex objects can be used for ordered visualization and so we need to subdivide user defined objects into convex objects. How to subdivide these user defined objects into convex object is an important issue. Many researchers have proposed many methods to generate the convex object from the concave object, we will give a brief introduction of our new convex object generation method. We know for 2DCCS, the convex object is polygon and for 3DCCS the convex object is polyhedron. In most works by others, they are subdivided into triangles and triangle mesh is used to represent the object. Actually, we need not do this, if a polygon is a concave, there must exist some concave voxel. Let us take 2DCCS as an example, there exist two kinds of concave voxel (CVX and CVY) as shown in Fig. 18.

In CVX voxel, we can take it as if we drag the voxel along the \( x \)-axis to eliminate the concavity; Similarly, we can drag the CVY voxel along the \( y \)-axis to eliminate it. If we can drag along either the \( x \) - or \( y \)-axis to eliminate the voxel, we still call it CVX because we use parallel ray scan along the \( x \)-axis to visualize the objects. If we can eliminate all the CVX and CVY, we will achieve our aim. From the property of concave voxel, we know if we extend the edge, which includes the concave voxel, the polygon will be cut into two parts, each of them will be located on one side of that edge. But to convex polygon, all edges will be located on the same side of any extended edge as shown in Fig. 19.

Our method is to extend every edge, which includes the concave voxel to cut the polygon into smaller parts. For convenience, we do not need to extend the edge because it will be difficult to calculate the intersection point; we can draw a line parallel to the \( x \)- or \( y \)-axis from the concave voxel to cut the polygon. For CVX, the line will be parallel to the \( x \)-axis; and for CVY, it will be parallel to the \( y \)-axis.

An example is outlined in Fig. 20.

This convex object generation method is more efficient than ordinary sort based method, which will subdivide
the object into triangle and trapezoid as outlined in Fig. 21.

Lin Feng [14] advanced a stack based convex object generation method outlined as Fig. 22.

The concave object generation method of Fig. 20 subdivides the object into three convex subdivision cells. Sort based method of Fig. 21 will generate six subdivision cells. Stack based method of Fig. 22 will generate five subdivision cells. Our method will generate fewer subdivision cells so that the list will be shorter to speed up the visualization procedure.

### 7. Results and discussion

In this section, we will give some results based on all ideas discussed in this paper. Before that we will use our method to evaluate the correctness of other space subdivision scheme. First, we will use our method to evaluate three space subdivision schemes: nonuniform space subdivision, uniform space subdivision and pyramid ray tracing. All these space subdivision schemes use ray based visualization instead of object based visualization. But according to what we have discussed in Section 5, we understand that we can evaluate them using the same method as object based visualization.

#### 7.1. Nonuniform space subdivision

There are many nonuniform space subdivision schemes. The main idea of nonuniform space subdivision is to use the image coherence to skip the empty areas to speed up the visualization procedure. Since octree is a typical nonuniform space subdivision scheme, we will take octree as an example. There are three kinds of subdivision cells in octree, empty cells, gray cells and full cells. All subdivision cells are cubes. An empty cell means all areas which it includes have no useful meanings and so it does not need to be subdivided further: A full cell means the area has been subdivided enough and does not need to be subdivided further. A gray cell means we are not clear about it and so it needs to be subdivided further if we want to see more detailed information. When we subdivide a cell, we will subdivide it into eight smaller subdivision cells; all eight cells are empty cells, gray cells or full cells. In octree, the object space is 3DCCS and the ray space is parallel ray. An example of octree is outlined in Fig. 23.

In Fig. 23, we use $S$ as the scene space. At the beginning, $S$ includes only one subdivision cell $A$ and it is a gray cell because we do not know what it includes. Because cube is a polyhedron, it is a convex space subdivision; at the same time the ray we use is parallel ray and so based on the knowledge of Sections 3–5 and Theorem 1, we know that it is an ordered subdivision.

For stage 1, we have

- $S_1 = \{A\}$, $|S_1| = 1$.

If we want to see more detailed information of scene space $S$, we need to subdivide it further. According to the subdivision scheme of octree, we only need to subdivide a gray cell into eight smaller cells; we denoted them as cell 1, 2, 3, 4, 5, 6, 7, 8. According to Section 5 and the idea of recursive space subdivision, we can replace cell $A$ by its descendent subdivision cells without losing the order. For stage 2, we have
This time, cell \( B \) is a gray cell while cell 13 is an empty cell and all others are full cells. We can further subdivide \( B \) into smaller cells if needed and denote them as 5, 6, 7, 8, 9, 10, 11, 12 and replace \( B \) with the new generated subdivision cells according to recursive subdivision idea.

Octree has the following criterion to stop the recursive subdivision procedure. Either all cells are empty cells or full cells or all cells have been subdivided enough. When the subdivision stops, the visualization procedure will only render the full and gray cells in the visualization chain and so octree can skip the empty areas to speed up the visualization procedure. No matter how many stages the octree needs to be subdivided, according to the previous proof, we know octree is a convex space subdivision scheme and can cause ordered visualization by parallel ray.

### 7.2. Uniform space subdivision

Uniform space subdivision is also an example of recursive space subdivision of convex space subdivision. It also subdivides a cell into eight smaller cells and every cell is a cube. The difference between octree and uniform space subdivision is that instead of only subdividing the gray cells, it will subdivide all cells in the same time and so we can look on it as a special case of octree subdivision by taking all cells as gray cells. The object space of uniform space subdivision is 3DCCS and the ray space consists of parallel rays. An example of uniform space subdivision is outlined in Fig. 24.

Hence, for Fig. 24, we use \( S \) to denote the scene space and we use parallel rays as the visualization ray. Originally, \( S \) includes only one subdivision cell \( A \). By now, it is the same as octree and so it is a convex space subdivision; since we use the parallel ray, based on the knowledge of Sections 3–5 and Theorem 1, we know that it is an ordered subdivision.

For stage 1, we have

- \( S_2 = S_1 = \{ A \} = \{ 1, 2, 3, 4, B, 13, 14, 15 \}, |S_2| = 8 \).

Now, if we want to see more detailed information of scene space \( S \), we need to subdivide it further. According to the rules of uniform space scheme, we need to subdivide all cells of the same level into eight smaller cells, which means, we need to subdivide \( A \) into eight smaller cells this time and we denoted them as cells \( B, C, D, E, F, G, H, I \), respectively. Since the new generated cells are all cubes, it is also a convex space subdivision. According to Section 5 and the idea of recursive space subdivision, we can replace cell \( A \) by its descendent subdivision cells without losing the order as octree.

For stage 2, we have

- \( S_3 = S_2 = \{ A \} = \{ 1, 2, 3, 4, B, 13, 14, 15 \} = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 \}, |S_3| = 15 \).
Furthermore, we can subdivide it again in order to obtain more detailed information. This time, we need to subdivide all the eight cells. We need to subdivide \( B \) into \( B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8 \) and \( C \) into \( C_1, \ldots, C_8 \). Finally, \( I \) into \( I_1, \ldots, I_8 \) accordingly. Now, according to the recursive subdivision idea, we replace every cell with its corresponding new generated subdivision cells.

- \( S_3 = S_2 = \{ B, C, D, E, F, G, H, I \} = \{ B_1, B_2, B_3, \ldots, B_8, C_1, \ldots, C_8, I_1, \ldots, I_8 \} \), \(|S_3| = 64\).

Uniform space subdivision scheme also needs to set a criterion to stop the recursive subdivision procedure. When the subdivision stops, the visualization procedure will render all subdivision cells in the visualization chain and so uniform space subdivision cannot skip the empty areas as octree. For uniform space subdivision scheme, at stage \( n \), the number of subdivision cells is \( 8^{(n-1)} \). From the above, we can make a conclusion that no matter how many stages the scene space needs to be subdivided, according to the previous proof, we know uniform space subdivision is a convex space subdivision scheme and can cause ordered visualization by parallel ray.

### 7.3. Pyramid ray tracing

Up to now, we have discussed and evaluated nonuniform space subdivision and uniform space subdivision schemes. The object spaces are 3DCCS while the subdivision cells are cubes and the ray space comprises parallel rays. Now, we will discuss a different space subdivision scheme, pyramid ray tracing. The object space of pyramid ray tracing is 3DPCS. Hence, we cannot use parallel rays to visualize it.

Fig. 25 is an example of pyramid ray tracing.

We denote the ray space of pyramid ray tracing as pyramid ray. Before the definition of pyramid rays, let us recall the knowledge of 3DPCS. From the definition of 3DPCS, we know the object space has an order \( R_{n} = \langle NAO_p, NAO_b, NAO_r \rangle \), with which we can enumerate the voxels. Now, we come to the definition of pyramid ray of 3DPCS,

- \( \forall r_1 \in R, \forall r_2 \in R, r_1 \neq r_2, \forall o_{p,k,t} \in r_1, \forall o_{r,u,v} \in r_2 \Rightarrow k \neq u \) or \( t \neq v \).
- \( \forall r_1 \in R, \forall o_{p,k,t} \in r_1, \forall o_{r,u,v} \in r_2, o_{p,k,t} \neq o_{r,u,v} \Rightarrow p \neq r \), \( k = u \), \( t = v \).

According to previous discussion, we know there exists an order \( R_o \) so that

- \( \exists R_o \), \( R_o \) is a linear order over the elements of rays of ray space,
- \( \forall r_1 \in R, \forall r_2 \in R, r_1 \neq r_2 \Rightarrow \text{either } r_1, r_2 \text{ or } r_2, r_1 \).

For pyramid rays, we have \( \forall r_1 \in R, \forall r_2 \in R, r_1 \neq r_2, \forall o_{p,k,t} \in r_1, \forall o_{r,u,v} \in r_2 \) according to definition of pyramid rays, \( k \neq u \) or \( t \neq v \), we define \( R_o \) as follows:

\[
\begin{align*}
& r_i R_o r_j, \quad k < u, \\
& r_i R_o r_j, \quad k = u, t < v, \\
& r_i R_o r_j, \quad k = u, t > v, \\
& r_i R_o r_j, \quad k > u.
\end{align*}
\]

We can extend the definition of \( R_o \) to the elements of every ray as follows:

- \( \forall r_1 \in R, \forall r_2 \in R, r_1 \neq r_2, r_1 R_o r_2, \forall o_k \in r_1, \forall o_i \in r_2 \Rightarrow o_k R_o o_i. \)

Now, we can define a new order \( R_t = \langle R_o, R_{t_o} \rangle \) so that we can use order \( R_t \) to enumerate the elements of every ray and every ray’s element.

From the discussion of volume visualization, we know the visualization is to find the intersection of scene space and ray space. Till now, we have used Cartesian scene spaces and use parallel rays to visualize it. This time, we can look on pyramid ray tracing as a subdividing ray space instead of scene space. For Fig. 25, at stage 1, we use only one ray cast from the eye to visualize the whole scene space and so we have

- \( R_1 = \{ A \}, |R_1| = 1. \)

If we want to obtain more detailed information, we can subdivide the ray into four subrays according to the definition of pyramid ray and so,

- \( R_2 = \{ A \} = \{ 1, B, 6, 7 \}|R_2| = 4. \)

Next, we can subdivide again; we take \( B \) as an example and we subdivide it into four more cells denoted as \( 2, 3, 4, 5 \), respectively. Using the recursive subdivision idea, we replace \( B \) with its decendent subdivision cells and so we have

- \( R_3 = \{ 1, 2, 3, 4, 5, 6, 7 \}|R_3| = 7. \)

Similar to the proof of Theorem 1, we can prove that pyramid ray tracing is also ordered visualization.

### 7.4. Object based visualization versus ray based visualization

We know object based visualization uses on object to intersect with rays while ray based visualization will use a ray to intersect with objects. So, we can understand that ray based visualization makes good use of the
coherence of ray space while object based visualization can make good use of the coherence of image space or scene space. Most of the time, the coherence of scene space is more important. Let us consider an example outlined in Fig. 26.

Hence for Fig. 26,
- \( S = \{ v_1, v_2, v_3 \} \) \( |S| = 3 \),
- \( v_1 = \{ o_{0,1}, o_{0,2}, o_{1,2} \} \) \( |v_1| = 3 \),
- \( v_2 = \{ o_{3,1}, o_{3,2}, o_{3,3}, o_{4,2}, o_{4,3} \} \) \( |v_2| = 5 \),
- \( v_3 = \{ o_{6,2}, o_{7,1}, o_{7,2}, o_{7,3} \} \) \( |v_3| = 4 \),
- \( |RS| = 10 \).

For ray based visualization, we use rays to intersect object and so we will get
- \( O_n = 10 \times (3 + 5 + 4) = 120 \).

Here, ray based visualization refers to visualization method that does not use scene space coherence as speedup scheme. (Octree is a visualization method using scene space coherence to skip the empty areas.)

In contrast, for object based visualization, we use objects to intersect rays and so we will get
- \( O_n = 3 + 5 + 4 = 12 \).

We may find octree is an exception of ray based visualization method because it can skip the empty area by using image space coherence. For octree,
- \( O_n = 3 + 5 + 4 = 12 \).

Hence, we may find some object based visualization and ray based visualization schemes can have the same time complex if both of them can make good use of scene space coherence. Besides, we can notice one advantage of object based visualization over octree. That is, instead of skipping only the empty areas, it can also skip areas of noninterest. In the case of octree, it only knows if a subdivision cell is empty or not while object based visualization also knows whether or not a subdivision cell is of interest to the user. The user can choose which part is of interest and which is of no interest and the visualization procedure can render them adaptively and interactively. Another advantage is that object based visualization uses a convex object to represent the scene space and so it is easy to integrate volume objects with graphics objects.

An overall comparison of the two visualization approaches is outlined in Table 1.

### 7.5. Results

We will present some results of our research. The dataset used is a CT head, MRI Head and 4D animated ultrasound heart. The object space is 3DCCS. In 3DCCS, we need to find the concave voxel, which is the 3D counterpart of concave pixel of 2DCCS. Further,
we will use planes parallel to the \( x \)-, \( y \)- or the \( z \)-axis in order to cut the concave object for generating the convex subdivision cells.

The dimension of the scene space is \( 256 \times 256 \times 256 \). We use parallel rays to do the visualization. In order to do object based visualization, we let the users define the object according to their knowledge. In this paper, we present a very simple demo to illustrate our ideas. The real application is far more complex than this one. In order to let the results be easily understood, we will analyze the result in two dimensions. The head, denoted as \( N_{0} \) will be subdivided into 1 and \( N_{1} \). \( N_{1} \) will be subdivided further into \( N_{2}, N_{3}, 4 \) and 7. \( N_{2} \) is subdivided into 2 and 3. \( N_{3} \) will be subdivided into 5 and 6. This is outlined as Fig. 27.

Most of these objects are not convex objects and so we have to subdivide them into convex objects. We take the right eye as an example, the subdivision is outlined in Fig. 28.

The final subdivision chain is shown in Fig. 29.

In the subdivision chain, the subdivision cells below the user level are transparent to the user because these subdivision cells are generated by our algorithm for implementation of our object based visualization method and these subdivision cells have no meanings to the users.

In our work, we can show the image adaptively according to the users’ request while performing the subdivision adaptively using the recursive subdivision method. Most of the time, since the users are only interested in specific objects in the scene space. We actually perform fast visualization to other parts with fewer subdivisions and give them a blur effect as background information. In our work, we manage the recursive subdivision and subdivision chain using a special data structure—segment tree, which will be introduced in another paper. In the following we present some results from our research.

Two different angle-viewing results are outlined in Fig. 30.

In Fig. 31, we show an example of an animated heart. We cut it into two objects and visualize them in the same scene space. Every object is an animated object and so the user can see both of them being animated. Furthermore, the user can manipulate them separately. The animated object is a 4D object, which means every object will change its appearance according to the time. In our research, we encapsulate all features of one object into a subdivision cell so that we can treat it as a normal subdivision cell. This is also an advantage of object based visualization. Just like integrating graphics object with volume object, we can also use subdivision cell to represent many different kinds of objects and insert it into the subdivision chain.

In Fig. 32, you can find that a head by CT scan has been subdivided into many meaningful parts and
visualized together in one scene space using object based visualization scheme.

In Fig. 33, an example of integrating graphics object and volume object into one scene space is given. Here, you can find in the same scene space, a half MRI head, a half CT head and a graphics ball. Using convex object based visualization method, we can show all these objects in correct order. With this, we can integrate computer generated virtual objects with the volume object gathered from 3D imaging devices in one space. For application, we can use it to integrate a virtual surgical tool with the volume rendered objects to supply a virtual surgical planning or augmented reality system.

We also prove that convex object based space subdivision is an ordered volume visualization scheme; using these ideas, we analyze some commonly used space subdivision approaches and speeding up techniques, such as uniform space subdivision and nonuniform space subdivision.

We also suggest a new convex object generation method, which can generate convex objects from concave scene space more efficiently. Some comparisons of our methods and other methods are also given. The subdivision cells generated by the convex object subdivision method are arbitrary convex polygons while those generated by sort based and stack based subdivision methods are triangles and trapezoids, hence we will get fewer subdivision cells, which can speed up the visualizing procedure.

In future, we will study fast regeneration method of the subdivision chain. This is required in application if the viewpoint is changed and thus requires the subdivision chain to be regenerated in order to do ordered visualization.

In this paper, we also highlighted the advantage of object based visualization. One important advantage of object based visualization is that we can fast rev visceral the objects. For example, if we do some modification to some objects such as rotating only the right eye in order to see the detailed information, according to the features object based visualization, we only need to rev visualize those objects that are affected by this operation.

8. Conclusion

In this paper, we provide the formal definitions for object space, space subdivision and ray visualization. We introduced some new concepts such as ordered space subdivision and ordered volume visualization. With these, we evaluated the correctness of a space subdivision approach of volume visualization.

References