ISBN 978 -987-633-574-4

Virtual Reality Volumetric Rendering Using Ray Marching with WebGL

Federico Marino[1], Horacio Abbate[1] and Ricardo A.Veiga[1],

[1] Universidad de Buenos Aires, Facultad de Ingeniería, Buenos Aires, Argentina
{fmarino, habbate, rveiga}@fi.uba.ar

Abstract.
This work is concerned with virtual reality web applications that implement volumetric visualizations based on Ray Marching, combined with traditional triangle-based rendering. The use case selected was an interactive tool to navigate through a 3D model of sedimentary basins and natural reservoirs of oil and gas. The surfaces are described by signed distance functions computed in the GPU within a pixel shader. A procedural 3D texture defines the distribution of sedimentary strata inside the volume and provides visual clues of the internal structures. In VR mode, the user can navigate the scene by moving his body. He can subtract portions of the 3D volume in real time through a set of Boolean operators (plane, cylinder or sphere) using VR hand controllers. A virtual control panel, dynamically generated from a JSON file, allows parameters to be adjusted within VR. The application can also be used in desktop mode.

Keywords: Ray Marching, Virtual Reality, Sedimentary Basin Visualization, Oil and Gas Industry, WebXR

1 Introduction

There are many uses for Virtual Reality (VR) systems in the gas and oil industry with the purpose of improving the efficiency and reducing the risks of exploration and operation [1]. Even though these techniques were widely applied by the industry for training, their uses were extended to other areas like exploring, production, and operation and maintenance of surface installations.

As Jampeisov [1] pointed out, a person receives up to 80% of the information of the world through sight and the use of 3D visualization helps to enhance the efficiency of analyzing large amounts of information. These ideas led to an increasing interest in using different computer-based technologies to train engineers [2].

The inclusion of computer and information technologies (TICs) in teaching methodologies is considered an alternative to improve the comprehension and use of scientific and technology models [3], [4], [5].

The present paper shows the implementation of a volumetric rendering solution based on ray marching that is compatible with WebGL 1.0 and WebXR and can run on a web browser. This project was financed by a PIDAE 2018 (1-116) grant.

1.1 The Problem to Solve

The structure of a seismic image of an oil basin is shown in Figure 1. This kind of schematic representation is commonly used and it consists in a sequence of bidimensional cuts. Each cut has a label that indicates the type of geological component (e.g.: the basin and its geometry, the filling represented by strata affected by geological faults).

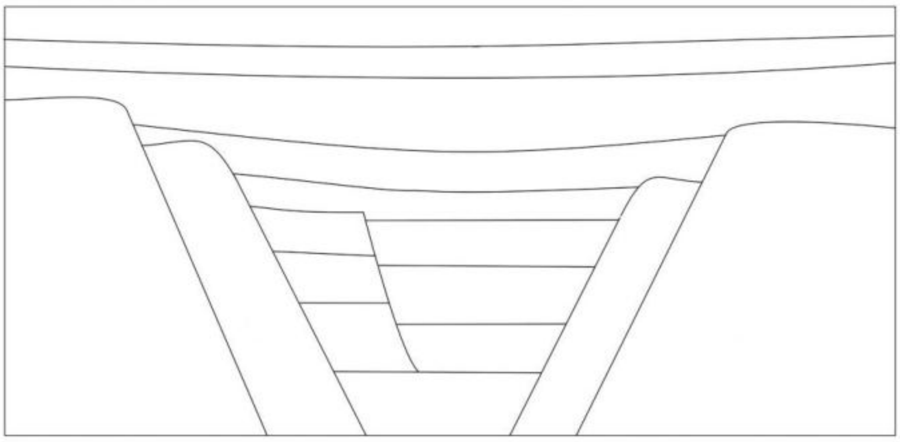

Fig. 1. Initial sketch of a simulated geological section for the RV model.

So far, the user had to analyze a two-dimensional scheme like the one shown in figure 1, in order to perceive the three-dimensional shape of each layer like porous and possibly permeable media. The VR application allows the user not only to see the bidimensional images, but also to get a tridimensional insight of the basin and to explore inside in a dynamic and intuitive way. This way the lithology, the textures and the sedimentary structures, and also the physical properties related to the context could be analyzed.

1.2 VR Devices

VR devices are more accessible to the general public nowadays. A high-resolution screen and an optical system provide an independent image to each eye for stereoscopic views. These systems also have a set of sensors to determine the attitude and the position of the user's head into the three-dimensional space. Many of them have wireless joysticks (hand controllers) whose orientation and position are also sensed.

In this project, an Oculus Quest device connected to a PC computer (which does the graphics processing) was used. The GPU used was a Nvidia RTX 2070.

ISBN 978 -987-633-574-4

1.3 WebGL, WebVR and WebXR Standards

The WebGL (Web Graphics Library) is a standard specification for an API implemented in Javascript for 3D graphics rendering by any web browser. It runs on any platform that supports OpenGL 2.0 and OpenGL 2.0 ES.

The WebVR standard was created by Mozilla in 2014, and it offers access to VR devices like HTC Vive, Oculus Rift, Oculus Quest or Google Cardboard from a web browser. The last version, launched in 2017, depends on special web browsers to run, like Firefox Nightly. The standard was never completely implemented and was finally substituted by the WebXR [6].

The WebXR standard brings together support for Virtual Reality and Augmented Reality applications on the same API (XR = AR + VR); as the devices for both applications have some common needs, like sensing the position and orientation, and also rendering the images from the corresponding point of view.

As of August 2021, WebXR is still a working draft and it's not available for all web browsers.

1.4 The Platform Chosen and the Solution Overview

A web application based on Javascript, HTML, CSS, WebGL and WebXR was implemented. Some open-source libraries such as React.js and Bootstrap were used to build the graphical user interface (2D menu).

Additionally, Three.js library was used for the management of 3D meshes, object hierarchies, cameras, light sources, materials, shaders and rendering.

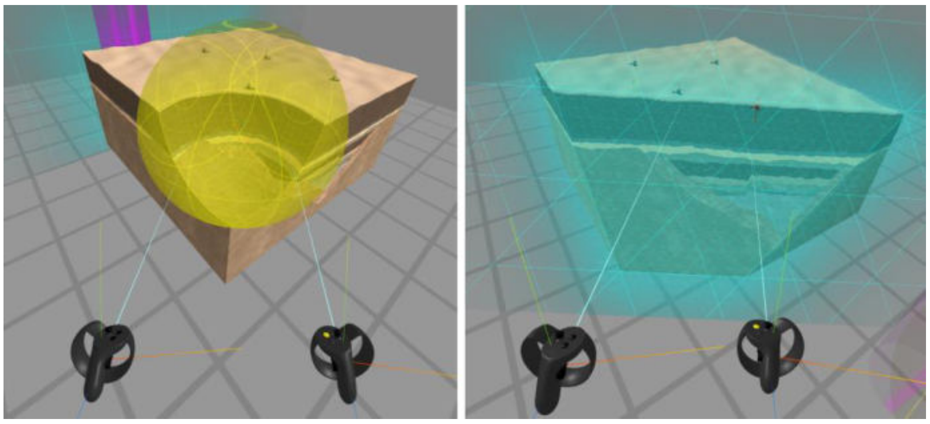

Fig. 2. Boolean operators being manipulated: sphere operator (left), flat operator (right).

The software lets the user navigate around a 3D volume that represents the basin. Using the hand controllers (as seen in figure 2), it is possible to position certain 3D objects called “Boolean operators” (sphere, plane or cylinder) that can interactively cut the volume, thus subtracting portions and allowing the user to visualize the interior of the basin.

ISBN 978 -987-633-574-4

2 The Rendering Engine

2.1 Representation of Surfaces

The problem of constructing a graphical representation of the basin can be divided into two parts, as shown in Figure 3: a) compute the coordinates of all the points that make up the surfaces resulting from subtracting the Boolean operators from the cube that represents the portion of terrain; b) compute the color of each pixel of the surfaces projected on the screen. To solve the first part, some way of modeling such surfaces dynamically must be considered, since the user can manipulate in real time the position, orientation and scale of Boolean operators.

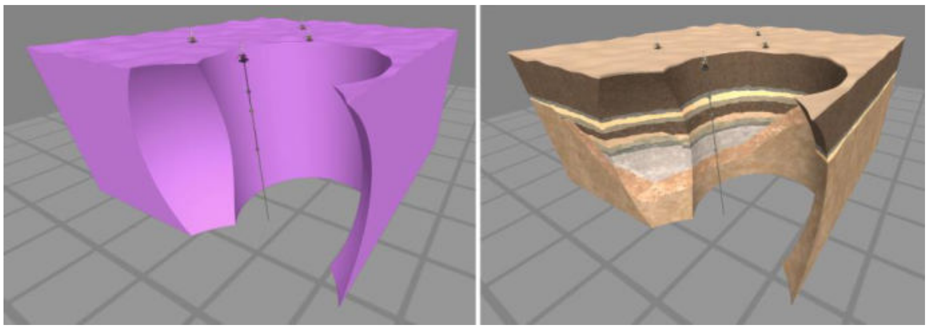

Fig. 3. Surface resulting from subtracting the spherical and cylindrical Boolean operators (left). Surface colored by a 3D procedural texture that exposes the internal structure of the oil reservoir. (right).

In computer graphics there are at least two ways of representing surfaces: the parametric form [7] and the implicit form [8]. The WebGL API allows the rendering of 3D scenes through a graphics pipeline that receives geometric information as input (vertices, triangles, normals, etc.) and generates an image on the output device.

In order to be able to visualize the surfaces, using the traditional graphical pipeline, it is necessary to obtain a mesh of triangles of the modeled surfaces.

The Marching Cubes technique [9] is an algorithm whose objective is to obtain a polygonal mesh of an iso-surface in a three-dimensional discrete scalar field (voxels). This technique is usually used for visualizing medical images taken by MRI scanners, where a density function is represented by values between 0 and 1.

In this case the density function could simply be computed from the Boolean operators and their position and orientation relative to the volume, assigning a density of 0 for an empty voxel and 1 for an occupied voxel.

Then, using Marching Cubes, it would be possible to get the triangle mesh that represents the surface shown in Figure 3.

One of the disadvantages of this approach is the processing time required to get the triangles, because this process should run every time any of the Boolean operators are modified. Although there are ways to implement it directly using a GPU, as described in GPU GEMS 3[10], it requires the use of geometry shaders, and this functionality is not available in the WebGL 1.0 platform.

2.2 Rendering of Implicit Surfaces Using Ray Marching

In this project, the ray marching technique was used for representing the surfaces of the basin. It is not based on triangles, but on evaluating each pixel in an image, based on implicit representations of 3D surfaces [7] described by Signed Distance Functions (SDF). These functions are analytic expressions representing distance fields, which measure how close a point *x* is to a set *S*, and its sign changes as it is on one side or the other of the set [11].

The 3D surfaces to be represented can be defined using multiple distance functions of primitive geometric shapes [12] (spheres, planes, cubes, cylinders, etc.), combined with operations like union, intersection, subtraction, etc.

This poses a challenge when it is needed to represent some specific arbitrary surface. In the particular case of this application, the primitives involved are few and simple (cubes, spheres, etc.).

This technique offers great advantages in terms of performance, since all the logic and data necessary to solve the scene can be embedded completely within a pixel shader program written in GLSL language; minimizing memory reads.

This program is supplied to the GPU and executed very efficiently by multiple processing cores working in parallel; solving an entire view in less than 10 milliseconds, with the appropriate hardware.

2.3 The Ray Marching Algorithm

The essence of the algorithm can be summarized as follows: for each pixel on the screen, a ray is defined by an origin (position of the camera or the observer's eye) and a direction vector towards the pixel in the near plane of the camera. Then, through an iterative process, the algorithm advances in that direction, evaluating at each step if the ray impacted any surface.

There are different variants of this algorithm; some of them, like the one proposed in [11], advance at a constant pace and therefore require a very high sampling rate on the path of the ray, which impacts on performance.

On the other hand, Hart [13] introduces a more robust method known as Sphere Tracing, where the advanced distance at each step is not constant, but depends on the evaluation of an SDF. If the resulting value *d* (distance to the surface) is positive, the position over the ray should be increased by *d*. If *d* is negative, the resulting position is within the implicit surface defined by the SDF.

The following code shows the essence of the algorithm, where *distanceToScene* is the actual SDF.

ISBN 978 -987-633-574-4

Ray marching function in GLSL

```
float raymarching(vec3 rayOrigin, vec3 rayDir) {
  float t=0.0;
  for(int i=0; i<MAX_STEPS ; i++) {
     float delta = distanceToScene(rayOrigin+rayDir*t);
     if (delta<epsilon)  return t;
     t+= delta;
  }
  return INFINITY; // max steps where reached
}
```

When the condition d < *epsilon* is met, the ray is considered to have hit the surface. Then the color of the pixel is evaluated at the point of impact using Phong's model [14]. The normal vector at the pixel can be obtained by calculating the gradient vector of the distance field, using 4 sampled points close to the impact point.

2.4 Procedural 3D Texture

The Phong model calculates the ambient (*ka*) and diffuse (*kd*) coefficients, based on the ambient light and a directional light representing the sun. Then, the diffuse color of the surface is the most relevant piece of information for the user, as it will help him recognize the different basin strata. The *getTerrainColor* function returns an RGB color based on two input parameters: *position* (vec3) and *normal* (vec3). A *colorMix* uniform variable controls whether the output should be a flat color or a pixel sampled from texture maps, as shown in figure 4.

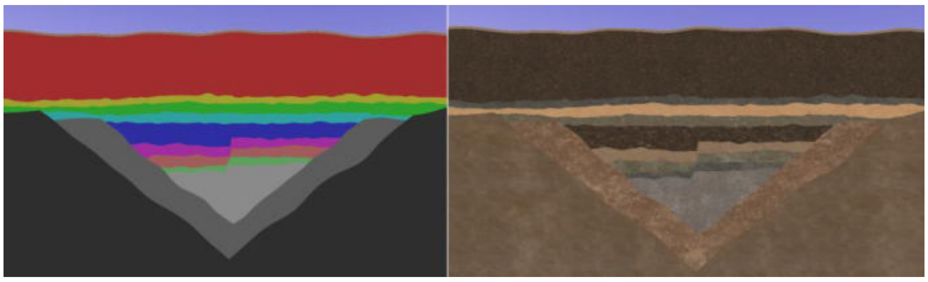

Fig. 4. Flat color output, with *colormix*=0 (left) texture color output with *colormix*=1 (right)

In this application there were 11 predefined types of basin strata, with their corresponding color and seamless texture map. For each component, a weight variable defines how much it contributes to the final pixel color.

In this model of an oil reservoir, most of the strata are horizontal layers with certain thickness. To compute the weight value, two *smoothstep* (ascending and descending slopes) functions are multiplied to generate a positive weight within a certain depth range. In order to get a more realistic appearance and to distort the straight-line borders between layers, a gradient noise [15] is applied on the *position* parameter, which represents the coordinates inside the basin.

2.5 Texture Mapping

In order to apply texture mapping to the raymarched surfaces, texture coordinates *(u,v)* need to be defined for every pixel. Considering that the surfaces are generated in real-time based on the result of the Boolean operators intersecting the volume, the texture coordinates also need to be generated dynamically. The triplanar mapping technique was chosen. This algorithm described by Geiss in [10], assigns *(u,v)* based on the one of the 3 cartesian planes XY, XZ or YZ whose normal is closer to the normal of the pixel, as shown in Figure 5. The *getTriplanar* GLSL function returns the texture map pixel based on the *position* & *normal* values at the pixel.

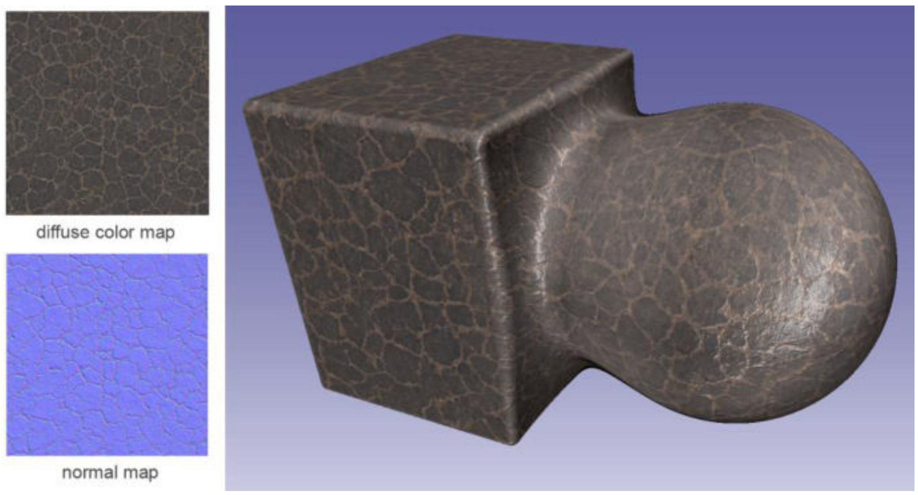

Fig. 5. The triplanar mapping algorithm applied to a dynamically generated surface.

2.6 Interactivity and User Experience

Once the VR mode is initiated, the user is immersed in a virtual scene consisting of an infinite extension floor and a colored cube of 1 meter on the side that floats at the height of the operator's line of sight. The user can walk around or can remain static and move to a different location in the virtual space, using the right controller thumbstick. By pointing his hand in the desired direction, the stick controls the speed of the motion.

ISBN 978 -987-633-574-4

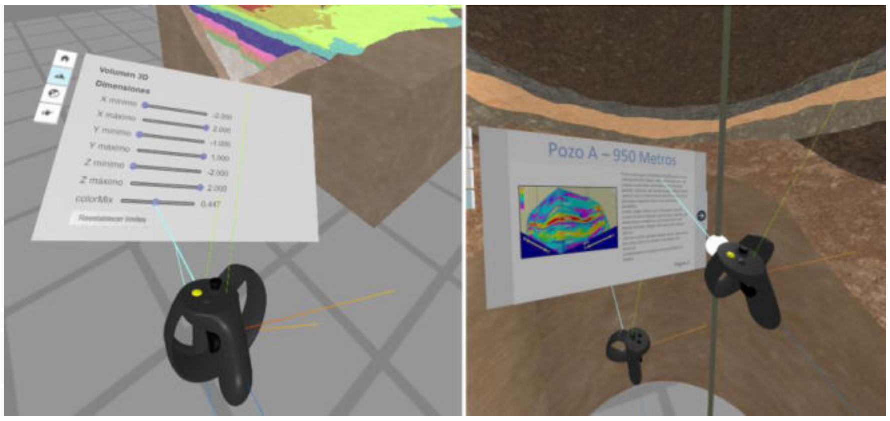

Fig. 6. Virtual control panel (left) drilling rigs “hyperlinks” (right).

The left-hand controller allows the user to show and hide a virtual control panel consisting of multiple tabs. Using the right controller, the user can point to and interact with different controls such as buttons and sliders as shown in Figure 6.

In addition, the hand controllers have side buttons associated with the grab and drop gesture, which allow the user to grab, rotate, move and drop the 3 types of Boolean operators, to interactively crop the 3D volume.

The 3D scene also features pre-designed 3D models that represent the drilling rigs and the oil well path. The user can point and select different targets along the well acting as "hyperlinks" that display related information on the panel (Figure 6 right). Finally, the user can take snapshots from his current point of view, which are automatically downloaded in JPG format, allowing a later analysis, outside of the virtual reality environment.

2.7 Virtual Control Panel

The application parameters can be controlled simultaneously from a 2D menu (on the browser window) or the virtual control panel in VR. The *MenuManager* component works as a centralized database for current application parameters. The 2D menu and the virtual control panel interact with it though getters & setters and *onValueChanged* events trigger when a parameter is modified.

The control panel is built dynamically from a JSON configuration file that defines the content of each tab, consisting of a list of controls (button, title, slider, switch or select). Each control mimics the behavior of their corresponding HTML counterparts but in 3D.

Three.js provides a class called *Raycaster* that can test the intersection of a ray (point + direction vectors) in the 3D scene, with a list of mesh objects. In some cases, invisible bounding box meshes are used, instead of the real mesh, to test the surface-ray collision (for example in the case of the sphere representing a slider’s cursor).

Every time the *Raycaster* detects a hit, it provides information on the object that was hit and the coordinates in world and local object space. The hand controller’s

ISBN 978 -987-633-574-4

location and orientation are provided by Three.js, through *VRController* class, which also provides event handlers for all the buttons and thumbsticks events.

2.8 Final Image Composition

The rendering function has two modes: desktop and VR. In VR mode, the viewport size is set to the dimensions of the VR device, provided by the WebXR API. In this mode, the viewport is divided in two halves (left & right eye view) and the rendering has to be executed twice on each frame using the corresponding view and projection matrices for each eye.

In order to generate a view from a camera, the rendering function requires two render passes. In the first pass, the ray marching algorithm is executed by a pixel shader that is assigned to a quad geometry matching the size of the viewport. The resulting image includes the basin volume, the floor and the background. This shader requires not only to output color information, but also depth for each pixel. This is important for compositing the next pass. In WebGL 1.0 it is not possible to output depth information from a fragment shader by default, so EXT_frag_depth needs to be enabled and available in the browser.

In the second pass all the objects that are represented by triangle meshes (Boolean operators, hand controllers, virtual control panel, oil rigs, etc.) are rendered. All of them are opaque with the exception of the Boolean operators that are translucent. As all transparent meshes are drawn at the end of the second pass, they blend properly with the previous content of the framebuffer.

3 Conclusions

The application developed allows the user to intuitively manipulate and explore a 3D volume, without the need to learn complex user interactions commonly used in 3D CAD software (using keyboard & mouse) thus reducing the learning curve for casual users.

The Boolean operators ease the study of the borders between strata and how they propagate inside a sedimentary basin. Moreover, the depth perception provided by the stereoscopic view, helps the user to properly interpret the shapes and scales.

The combination of ray marching and scanline rendering techniques, and the implementation of a procedural 3D texture, allowed the efficient implementation of a real-time application with low latency and a good performance, without the uncomfortable effects usually associated with virtual reality experiences.

The WebXR API, while still needs improvements, seems to be a promising alternative to build and deploy multi-platform VR experiences to wide audiences, at lower costs and reduced delivery times.

ISBN 978 -987-633-574-4

References

1. Jampeisov, Z.: Using Virtual Reality Technology in Oil and Gas Industry. 9, 124–127 (2019). https://doi.org/10.31033/ijemr.9.2.15.
2. Aveleyra, E.E.: Aportes para el debate: Las tecnologías en la enseñanza universitaria: nuevos escenarios, nuevos desafíos. En C. Nosiglia (comp.). La Universidad de Buenos Aires. Aportes para la CRES, pp.177-189. Editorial Universitaria de Buenos Aires, Buenos Aires (2018).
3. Cabero Almenara, J., Barroso Osuna, J.: Nuevos retos en tecnología educativa. Síntesis, Madrid (2015).
4. Coll, C., Monereo, C.: Psicología de la educación virtual. Aprender y enseñar con las tecnologías de la información y comunicación. Morata, Madrid (2008).
5. Martí, R., Gisbert, M., Larraz, V.: Ecosistemas tecnológicos de aprendizaje y gestión educativa. Características estratégicas para un diseño eficiente. Edutec. Revista Electrónica de Tecnología Educativa, (64), pp. 1-17 (2018).
6. WebXR Standard, https://www.w3.org/TR/webxr.
7. Hughes, J. F., van Dam, A., McGuire, M., Sklar, D. F., Foley, J. D., Feiner, S. K., Akeley, K.: Computer Graphics - Principles and Practice, 3rd Edition. Addison-Wesley (2014).
8. Menon, J.: An Introduction to Implicit Techniques. ACM SIGGRAPH 96 Course Notes: Implicit Surfaces for Geometric Modeling and Computer Graphics, pages A1–13 (1996).
9. Newman, T. S., Yi, H.: A survey of the marching cubes algorithm. 30, 854–879 (2006). https://doi.org/10.1016/j.cag.2006.07.021.
10. Nguyen, H.: GPU gems. Addison-Wesley, Upper Saddle River, NJ [u.a.] (2007).
11. Rendering Worlds with Two Triangles with raytracing on the GPU in 4096 bytes. https://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf.
12. Modeling with distance functions, https://iquilezles.org/www/articles/distfunctions/distfunctions.htm.
13. Hart, J., C.: Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. The Visual Computer, 12(10):527–545, (1996).
14. Phong, B.: Illumination of Computer-Generated Images, Department of Computer Science, University of Utah, UTEC-CSs-73-129 (1973).
15. Ebert, D., Musgrave, K., Peachey, D., Perlin, K., Worley. Texturing and Modeling: A Procedural Approach. Academic Press, October 1994. ISBN 0-12-228760-6