Department of Electrical Engineering
State University of New York at Stony Brook
Stony Brook, New York 11794-2350

Stony Brook
Computer Vision Laboratory

Tech. Report 93.04.21 April 21, 1993

Computer Modeling and Simulation of an Active Vision Camera System

Ming-Chin Lu and Muralidhara Subbarao

Abstract

Verification of computer vision theories is facilitated by the development and implementation of computer simulation systems. Computer simulation avoids the necessity of building actual systems; they are fast, flexible, and can be easily duplicated for use by others. We proposed, in our previous work, an useful computational model to explore the image sensing process. This model decouples the photometric information and the geometric information of objects in the scene. In this report, we extend the proposed image sensing model to simulate the formation of moving objects and stereo imaging applications. The simulation algorithms for curved objects, motion simulation, and stereo imaging are presented. Based on the proposed model and algorithms, a computer simulation system called Active Vision Simulator (AVS) have been implemented. The implementations are efficient, modular, extensible, and user-friendly so that others can easily reproduce and/or verify their experiments on a broader set of computer vision theories.

Computer Modeling and Simulation of an Active Vision Camera System

Ming-Chin Lu and Muralidhara Subbarao
Department of Electrical Engineering
State University of New York
Stony Brook, NY 11794-2350

Abstract

Verification of computer vision theories is facilitated by the development and implementation of computer simulation systems. Computer simulation avoids the necessity of building actual systems; they are fast, flexible, and can be easily duplicated for use by others. We proposed, in our previous work, an useful computational model to explore the image sensing process. This model decouples the photometric information and the geometric information of objects in the scene. In this report, we extend the proposed image sensing model to simulate the formation of moving objects and stereo imaging applications. The simulation algorithms for curved objects, motion simulation, and stereo imaging are presented. Based on the proposed model and algorithms, a computer simulation system called Active Vision Simulator (AVS) have been implemented. The implementations are efficient, modular, extensible, and user-friendly so that others can easily reproduce and/or verify their experiments on a broader set of computer vision theories.

1 Introduction

Many theories have been developed in computer vision during the past three decades for recovering scene information. Verification of such computer vision theories often require expensive and accurate camera systems, and laboratory facilities for calibration and experimentation. As an alternative, it is possible to develop computational models of the camera system, and simulate the system on a computer. This is not only faster and cheaper than building actual camera systems and setting up expensive laboratories, it also provides flexibility and accuracy. The physical parameters of the camera system (*e.g.* focal length, sampling rate, quantization level, noise characteristics, optical aberrations, etc.) are easily changed and they can be set to desired values to very high accuracy. The only major limitation is the amount of computational resources required for simulation.

In active vision, changing the direction of view and the visual parameters facilitates and makes efficient the computational stage of machine vision. An active vision system can be considered as a system that integrates visual sensing and action. There are two common tasks to be solved in active vision systems: one is the correspondence problem in stereo imaging, the other is motion estimation to dynamically track the objects in the scene. Many researchers have proposed algorithms [1, 2, 3, 4, 5, 6, 8, 11] for these tasks. Our objective here is to provide researchers a simulation environment to simulate image sensing process in motion and stereo systems.

In our previous work[7, 10], we proposed a computational model on the image sensing and formation process of a CCD camera system. This model decouples the photometric properties of a scene from the geometric properties of the scene in the input to the camera system. In this report, we further extended the proposed computational model to simulate the image formation of moving objects (motion) and stereo vision system. Based on the extended computational model, a computer simulation system called Active Vision Simulator (AVS) is developed. AVS is an extension of the Image Defocus Simulator (IDS) presented in[7, 10]. It can be used to simulate image formation process in a monocular (MONO mode) or a binocular (STEREO mode) camera system. The simulation of curved objects is also included in AVS. The user interfaces for AVS are similar to those in IDS, *i.e.,* two graphical user interfaces – Sunview Graphical Interface (SGI) and X-window Graphical Interface (XGI), and a dummy terminal user interface – Dummy TTY Interface (DTI). Finally, a depth map generation program using ray casting algorithm is also included in AVS as a tool for curved object simulation.

This report is organized as follows: Section 2 presents the computational model for motion and stereo simulation; Section 3 describes the simulation algorithms used for curved objects, motion simulation, and stereo imaging; Section 4 describes the user interfaces of AVS; Section 5 presents the simulation results; and finally, Section 6 concludes this report.

2 Camera Model

In this section, we will extend the computational model presented in[10] to simulate the image sensing process for moving objects and binocular stereo camera systems.

2.1 Motion Simulation

When objects move in front of a camera, or when a camera moves through a fixed environment, there are corresponding changes in the images. The displacement of a point in the environment will cause a displacement of the corresponding image point. In motion simulation, we assume that all the objects in the scene are rigid objects. Therefore, the shape of the objects will not change during motion.

Figure 1 shows the relationship between an object motion vector $\vec{m}_o = \vec{P_oP'_o} = [V_{x0} \quad V_{y0} \quad V_{z0} \quad \Delta t]$ and the image motion vector $\vec{m}_i = \vec{P_iP'_i}$. For simplifying the discussion, the image plane is placed at the focused position and is perpendicular to the optical axis (z-axis). The vector $\vec{m}_o$ can be decomposed into two components, one parallel to the x-y plane ($\vec{P_oP''_o}$) which shifts the object, and another parallel to the z-axis which changes the size of the object.

Consider the translation vector $\vec{P_oP''_o}$. Let $\vec{P_oP''_o} = [\vec{V_t} \quad \Delta t] = [V_{xo} \quad V_{yo} \quad 0 \quad \Delta t]$ for a fixed time interval Δt. This corresponds to a motion vector $\vec{m}_i = [\vec{V_i} \quad \Delta t] = [V_{xi} \quad V_{yi} \quad 0 \quad \Delta t]$ in the image plane. The amount of displacement is $||\vec{P_oP''_o}|| = ||\vec{V_t}\Delta t||$ in the scene which corresponds to a displacement of $||\vec{P_iP'_i}|| = ||\vec{V_i}\Delta t||$ in the image plane.

From the geometry in Figure 1, we have

$$\frac{||\vec{V_i}\Delta t||}{||\vec{V_t}\Delta t||} = \frac{r_i}{r_o} = \frac{v - f}{f} \tag{1}$$

The displacement of points in the image plane can be computed using Equation (1). For z-axis movement, *i.e.* $V_{z0} \neq 0$, there will be a change in the size of the objects in the scene. This results in image magnification or shrinking which requires image interpolation and resampling.

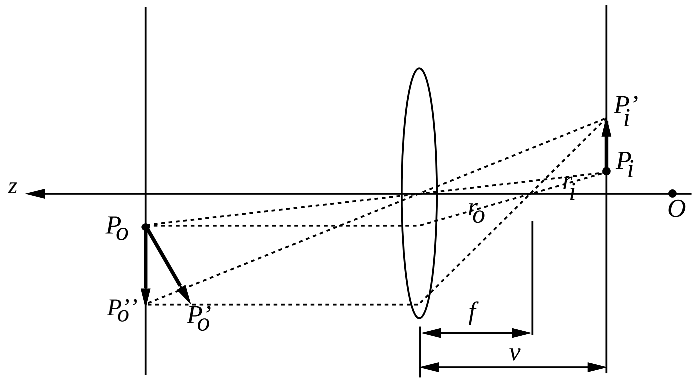


Figure 1: Relationship between the displacement of a point in the scene and the corresponding point in the image plane.

2.2 Stereo Vision System

A general stereo system model is shown in Figure 2 where O is the global origin, O' and O'' are the entrance pupil origin of the left and the right cameras, respectively. The left and the right cameras can be treated as monocular camera systems similar to that in Figure 3. The global origin O is introduced as a reference point for the positions of the object, the left camera, and the right camera.

In this generalized stereo system, the optical axes of the two cameras are not parallel, but intersect at some point in the scene. In order to simplify computations in our simulation, we restrict the optical axes of the cameras to be parallel. Therefore, in this model, there is a relative translation between the two cameras, but no relative rotation. This restriction can be removed at the expense of more computation.

Figure 4 is the global coordinate system used in our current stereo simulation where z-, z'-, and z''-axes are parallel to each other. Based on this configuration, the stereo vision system can be modeled as shown in Figure 5. The scene information is first translated, and scaled with respect to the origin of each camera. After this transformation, the photometric information $f(\theta, \phi, \lambda, t)$ and the

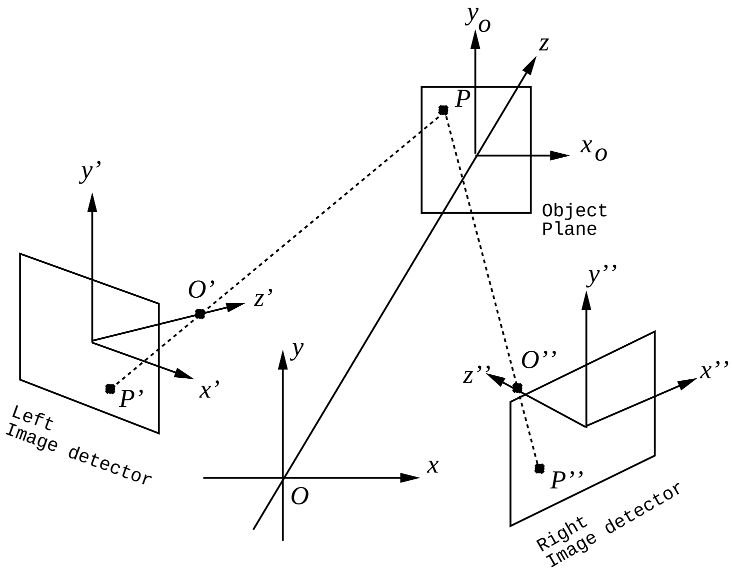


Figure 2: A general stereo system model.

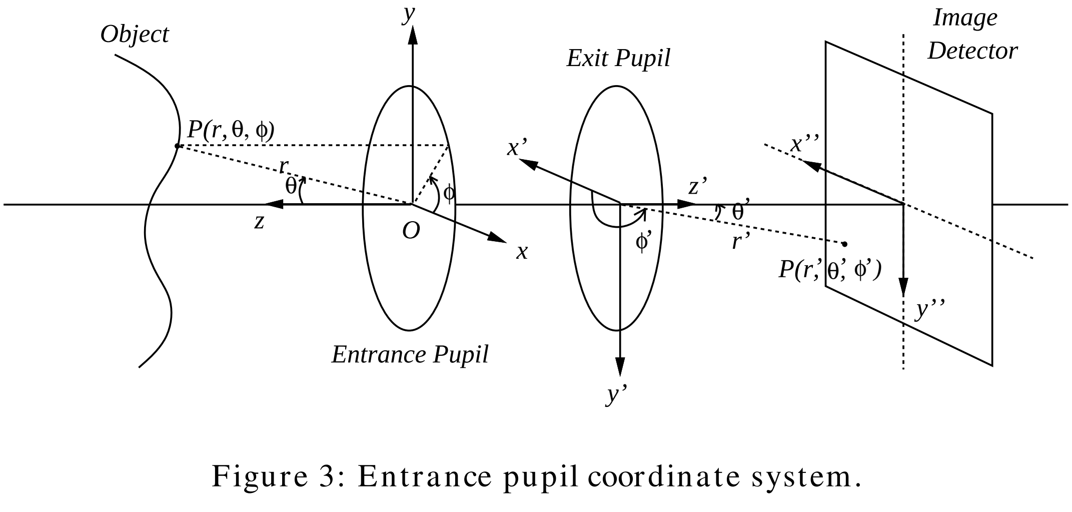


Figure 3: Entrance pupil coordinate system.

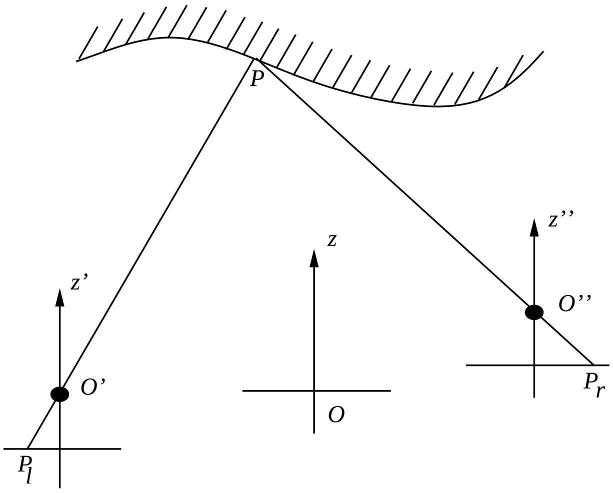


Figure 4: Global coordinate system used in stereo simulation.

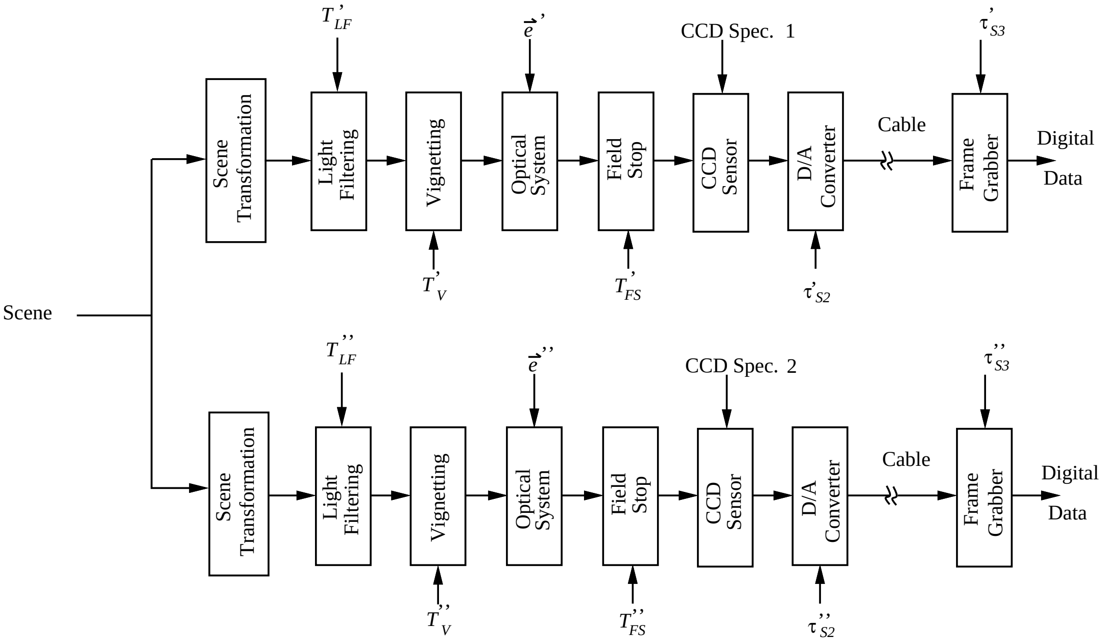


Figure 5: Block diagram of a stereo vision system.

geometric/depth information $r(\theta,\phi)$ are transformed to $f_l(\theta,\phi,\lambda,t)$, $f_r(\theta,\phi,\lambda,t)$ and $r_l(\theta,\phi)$, $r_r(\theta,\phi)$ for the left and the right cameras. These functions are the input to the camera system. The remaining functional blocks are the same as those presented in[10].

3 Simulation Algorithms

3.1 Curved Objects

Consider the photometric information $f(\theta,\phi,\lambda,t)$ and the geometric information $r(\theta,\phi)$. $r(\theta,\phi)$ contains the depth information of objects in the scene. For curved objects, $r(\theta,\phi)$ can not be approximated by a constant u. Under this situation, the point spread function is space-variant and is specified by $h(\theta,\phi,\theta',\phi',r(\theta,\phi),\vec{e})$ as discussed in[10]. In a Cartesian coordinate system, the geometric information and the point spread function can be represented as $r(x,y)$ and $h'(x,y,r(x,y),\vec{e})$, respectively, under the assumption that all CCD elements have the same characteristics. In this case, the output of the optical system will be:

$$f_3(x,y,\lambda,t) = h'(x,y,r(x,y),\vec{e}) \star f_2'(x,y,\lambda,t) \tag{2}$$

where $\star$ is the convolution operator.

Assume there are N different distances $(r_i, i = 1,\cdots,N)$ in the scene. Using superposition, $f_2'(x,y,\lambda,t)$ can be decomposed into N components as:

$$f_2'(x,y,\lambda,t) = \sum_{i=1}^{N} f_{2i}(x,y,\lambda,t) \tag{3}$$

where

$$f_{2i}(x,y,\lambda,t) = \begin{cases} f_2'(x,y,\lambda,t), & \text{if } r(x,y) = r_i \\ 0, & \text{elsewhere} \end{cases}$$

Step 1: Decompose the object into N planes, $f_{2i}(x, y, \lambda, t)$, of distance
$r_i, i = 1, \cdots, N$, according to the depth map information;
Step 2: **for** $i = 1$ **to** N **do**
begin
Compute and store the point spread function h_i
end;
Step 3: $f_3 \leftarrow 0$;
for $i = 1$ **to** N **do**
begin
$f_3 \leftarrow f_3 + f_{2i} \star h_i$
end;

Figure 6: Simulation algorithm for curved objects.

Thus, Equation (2) becomes:

$$\begin{aligned} f_3(x, y, \lambda, t) &= \sum_{i=1}^{N} h'(x, y, r(x, y), \vec{e}) \star f_{2i}(x, y, \lambda, t) \\ &= \sum_{i=1}^{N} h_i(x, y, \vec{e}) \star f_{2i}(x, y, \lambda, t) \end{aligned} \tag{4}$$

where $h_i(\cdot)$ is the point spread function for the planar object at distance r_i. Note that, if the profile of the scene in a small field-of-view is smooth, we have $N = 1$ and

$$f_3(x, y, \lambda, t) = h'(x, y, \vec{e}) \star f_2'(x, y, \lambda, t)$$

as derived in[10]. Therefore, the algorithm for the simulation of curved objects can be summarized as in Figure 6 where FFT algorithm can be applied in Step 3 to reduce the large amount of computations needed.

The depth map information can be obtained from, *e.g.*, range scanner or the ray casting algorithm presented in Appendix Appendix A.

3.2 Motion Simulation

The motion parameters used in the simulation are specified by the vector $\vec{m} = [V_x \quad V_y \quad V_z \quad \Delta t]$, where V_x, V_y, and V_z are the velocity components of the motion; Δt is the duration of the motion. Here, we assume that the scene contains only rigid objects so that the object will not change its shape while it is moving.

For objects moving perpendicular to the optical axis, *i.e.* $V_z = 0$, the size of the objects in the scene will remain unchanged. However, part of the original image will move out of the field-of-view and will not appear in the image plane. This will also introduce other objects into the scene which are not in the original image. Therefore, the original input image must include the objects that may come into the camera's field of view due to motion. This problem can be avoided by assuming a dark background. When parts of the objects move out of the camera's field of view, the dark background appears in the field of view. Here, we use this approach for its simplicity and efficiency in memory management.

When an object moves toward or away from the camera, the objects in the scene will be enlarged or shrunk. Therefore, resampling must be done to compensate for this effect. In AVS, we use bi-linear interpolation to compute the value $g(m, n)$ from its four neighbors $f(i, j)$, $f(i+1, j)$, $f(i, j+1)$, and $f(i+1, j+1)$. The result is:

$$g(m, n) = a \cdot (m - i) + b \cdot (n - j) + c \cdot (m - i)(n - j) + d \tag{5}$$

where $i \leq m \leq i + 1$, $j \leq n \leq j + 1$, and

$$\begin{aligned} a &= f(i + 1, j) - f(i, j) \\ b &= f(i, j + 1) - f(i, j) \\ c &= f(i + 1, j + 1) + f(i, j) - f(i, j + 1) - f(i + 1, j) \\ d &= f(i, j) \end{aligned}$$

Step 1: **if** $V_x \neq 0$ or $V_y \neq 0$ **then**
begin
Shift the object horizontally by the amount $V_x \Delta t$;
Shift the object vertically by the amount $V_y \Delta t$
Append dark background if part of the object is
moved out of the view;
end;
Step 2: **if** $V_z \neq 0$ **then**
begin
if $V_z > 0$ **then**
move the object toward the camera and resample;
else /* $V_z < 0$ */
move the object away from the camera and down-sample;
Append dark background if the neighbor of the object
appears in the view;
end;

Figure 7: Simulation algorithm for moving objects.

The simulation of an object moving with an arbitrary motion vector $\vec{m}$ is done by a shift operation if $V_x \neq 0$ or $V_y \neq 0$, and then a resampling operation if $V_z \neq 0$ to get the synthesized image. The algorithm is shown in Figure 7. Note that, during up-sampling process, the image might be smoothed, while in the down-sampling process, some image details might be lost.

3.3 Stereo System

For a binocular camera system, the two camera positions are specified by the vectors $\vec{O}_l = [x_l \quad y_l \quad z_l \quad \theta_{xl} \quad \theta_{yl} \quad \theta_{zl}]$ and $\vec{O}_r = [x_r \quad y_r \quad z_r \quad \theta_{xr} \quad \theta_{yr} \quad \theta_{zr}]$ with respect to the global origin O in Figure 2. The components of these vectors specify the positions and the orientations of the two cameras.

Stereo image pairs can be generated using the motion algorithm presented

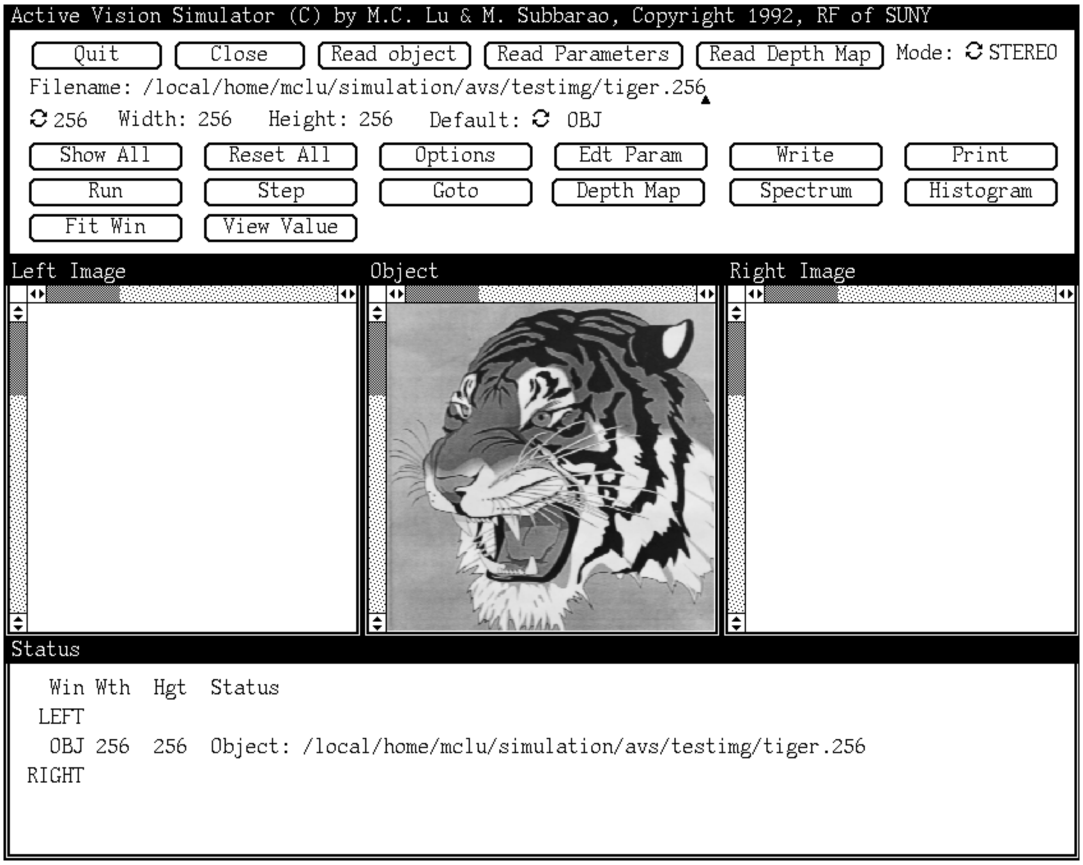


Figure 8: AVS graphical user interface.

in Figure 7 where the motion displacement corresponds to $(-x_l, -y_l, -z_l)$ for the left camera and $(-x_r, -y_r, -z_r)$ for the right camera. Note that, the orientation parameters are fixed to be $\theta_x = 90°$, $\theta_y = 90°$, and $\theta_z = 0°$.

4 The User Interfaces

Three user interfaces are provided in AVS – SGI, XGI, and DTI. The appearance and the basic functions of these user interfaces are similar to those in IDS. AVS has all the functions of IDS plus one more window and some additional features as shown in Figure 8. Besides, the single parameter window in IDS is now three

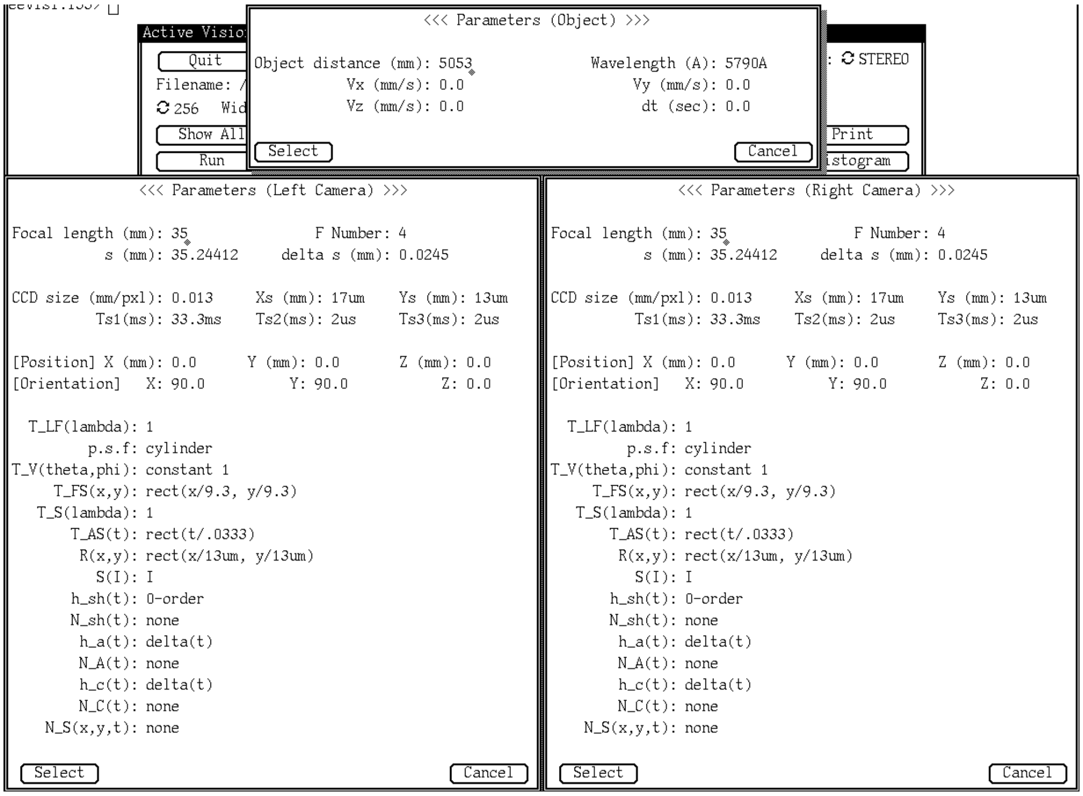


Figure 9: Categorized parameters in AVS.

parameter windows in AVS – one each for the left and the right camera (camera parameters), the other for object-specific parameters as shown in Figure 9

For curved object simulation, the depth map is read from a file by using the "Read DepthMap" command. The depth information stored in the file is a relative value, $\Delta r(x, y)$, with respect to the object distance u which is the shortest distance between the global origin O and the scene (*i.e.*, $\min\{d_i, i = 1, \cdots, N\}$). The object distance $r(x, y)$ is computed as

$$d(x, y) = \Delta r(x, y) \cdot k + u$$

where k is the scaling factor option in the option menu popped up by the "Option" command. The format of the depth map file is also specified in this menu.

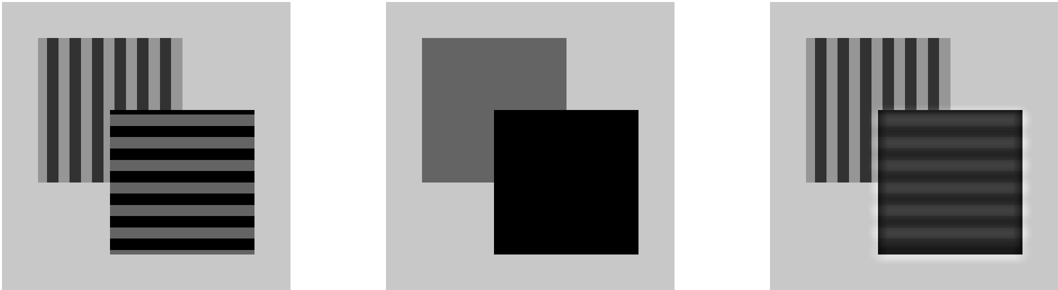
When the depth map is loaded, depth information $r(x, y)$ can be viewed by using the “Depth Map” command which will pop up a window with depth profile. The value of the depth at each point can then be viewed on the screen by moving the mouse pointer to the desired location. In DTI, the value is displayed according to the command line arguments used.

The parameters can be edited/viewed by the “Edt Param” command which searches the “Default” field for target window. The target can be object parameters, left camera parameters, or the right camera parameters as shown in Figure 9. The object parameters contain object distance and wave length information for general object information; and V_x, V_y, V_z, dt for motion information. The camera parameters are basically the same as those in IDS except that (i) the object distance and wave length information are moved to the object parameters window; and (ii) the camera position and orientation information ($\vec{O}_r$, $\vec{O}_l$) are added.

Another added feature is the “Mode” choice in Figure 8 which can be toggled between MONO and STEREO mode to simulate monocular and binocular image formation process. In “MONO” mode, “Left Camera” window will disappear. Therefore, the image will be synthesized in the “right camera” window by default. All the other commands are borrowed from IDS and carry the same functions. Besides, the 3-D object generation program described in Appendix Appendix A is also integrated into this system as a tool to generate the depth map information $r(x, y)$.

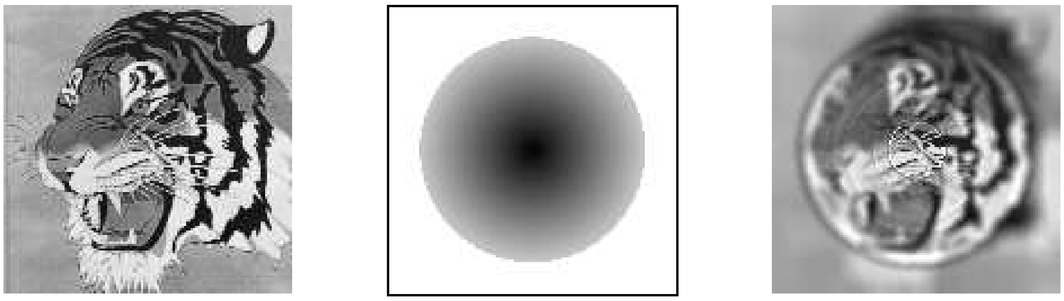
5 Simulation Results

In this section, some simulation results are presented to illustrate the capability of AVS simulator.



(a) (b) (c)

Figure 10: Simulated images for two planar boxes placed at different distances.



(a) (b) (c)

Figure 11: Simulated images for object placed on a cone-shaped depth map.

5.1 Curved Objects

Figure 10 gives a simulated image of two striped boxes placed at two distances. The scene and the depth map are shown in Figure 10(a) and Figure 10(b), respectively. Note that, the darker the value the depth map is coded, the closer the object is to the camera. The horizontal-striped box is located near the camera, while the vertical-striped box is located away from the camera. The camera parameters are adjusted to focus at the vertical-striped box. The resulting image is shown in Figure 10(C).

(a) $\vec{m}_1$ (b) (c) $\vec{m}_2$

Figure 12: Resampled images in motion simulation.

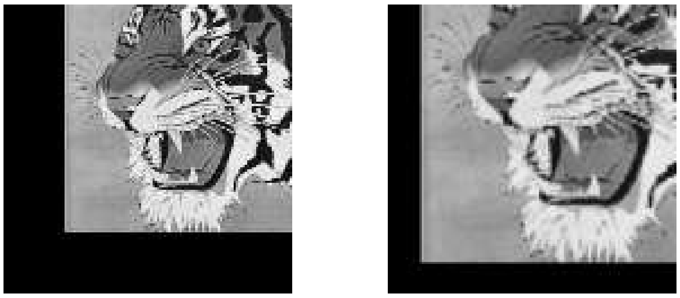
Another example is the tiger face placed on a cone-shape depth map as shown in Figure 11(a) and Figure 11(b), respectively. The depth range is from 2000mm to 3600mm (inside the cone) and the camera parameters are adjusted to focus at an object distance of 2000mm. The resulting image is shown in Figure 11(c). Note that, the depth outside the cone (white area) is assumed to be infinity. Therefore, a circle is visible in Figure 11(c).

5.2 Motion

Figure 12 shows the simulated images of moving objects. The center image is the original one. The left and the right images are generated with motion vector $\vec{m}_1 = [0 \quad 0 \quad -2.5m/s \quad 1sec]$ and $\vec{m}_2 = [0 \quad 0 \quad 2.5m/s \quad 1sec]$, respectively. All other parameters are the default ones in Figure 9.

The simulation of the shift operation (motion with $V_z = 0$) and the combined operation are shown in Figure 13 with $\vec{m}_3 = [100 \quad 100 \quad 0 \quad 1sec]$ and $\vec{m}_4 = [100 \quad 100 \quad 2.5m/s \quad 1sec]$. All other parameters remain the default ones.

Note that in Figure 12(c), Figure 13(a) and (b), dark background is intro-



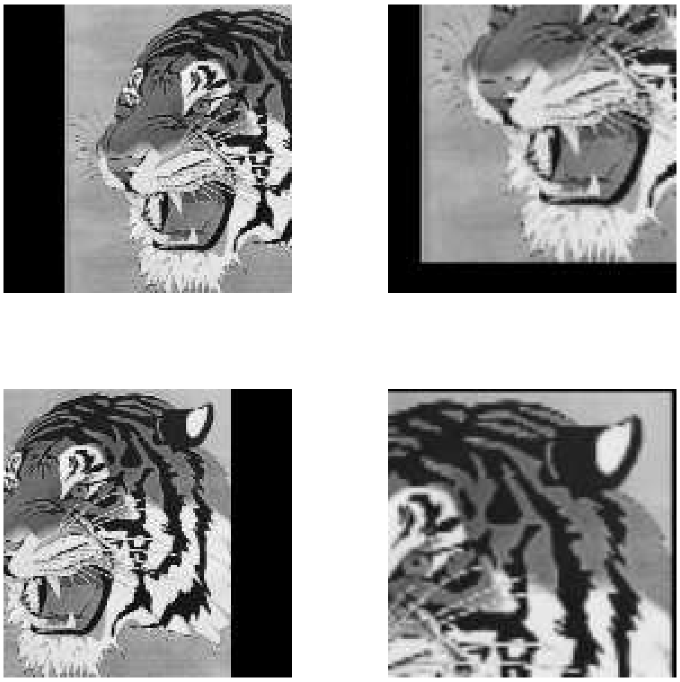
(a) $\vec{m}_3$ (b) $\vec{m}_4$

Figure 13: Simulated images under shift operation and the general motion vector.

duced because the object is moved away from camera or part of the object move out of the field of view as mentioned in Section 3.

5.3 Stereo

The simulation of the stereo image pairs for the left camera position $\vec{O}_l = [-100mm \quad y_l \quad z_l \quad 90° \quad 90° \quad 0]$ and the right camera position $\vec{O}_r = [100mm \quad y_r \quad z_r \quad 90° \quad 90° \quad 0]$ is shown in Figure 14 where the first row is the image on the left camera, the second row is the image on the right camera. In Figure 14(a), $y_l = z_l = y_r = z_r = 0$ which corresponds to the shift operation; in Figure 14(b), the left lens is moved toward the object with $y_l = -100mm, z_l = 500mm$ while the right camera is moved toward the camera with $y_l = 100mm, z_l = -500mm$. The image resampling and the dark background effect are visible in these simulations.



(a) (b)

Figure 14: Simulation of stereo image pairs.

6 Conclusion

In this report, we have implemented the curved object, motion, and stereo image sensing simulation in the computer simulation package called Active Vision Simulator. AVS is a natural extension of IDS presented in the previous work[10]. It can be used to synthesize the images for research on image restoration, motion analysis, depth from defocus, and algorithms for solving the correspondence problem in stereo vision area.

The efforts spent on extending the IDS to AVS is limited – two added modules on motion and stereo, and some changes in the user-interfaces – because of the module design and embedded extensibility of our original design of IDS. Again, AVS can also be easily extended if needed and can be used by other researchers on the verification of various vision theories.

References

[1] S. T. Barnard and M. A. Fischler, "Computational Stereo," *Computing Surveys*, Vol. 14, No. 4, Dec. 1982.

[2] P. Bouthemy and P. Lalande, "Motion Detection in an Image Sequence Using Gibbs Distributions," *Proceeding of IEEE International Conference on Acoustic, Speech, and Signal Processing,* pp. 1651-1654, May 1989.

[3] B. K. P. Horn, *Robot Vision,* McGraw-Hill, New York, 1986.

[4] J. Konrad and E. Dubois, "Bayesian Estimation of Motion Vector Fields," *IEEE Transaction on Pattern Analysis and Machine Intelligence,* Vol. 14, No. 9, pp. 910-927, Sep. 1992.

[5] E. P. Krotkov, *Exploratory Visual Sensing for Determing Spatial Layout with an Agile Stereo Camera System,* Ph.D. dissertation, Department of Computer and Information Science, University of Pennsylvania, PA, April 1987.

[6] Y. Liu and T. S. Huang, "A Linear Algorithm for Determining Motion and Structure From Line Correspondences," *Computer Vision, Graphics, and Image Processing,* Vol. 44, No. 1, pp. 35-57, 1988.

[7] M.-C. Lu and M. Subbarao, "Image Defocus Simulator: A Software Tool," Technical Report No. 92.05.27, Computer Vision Laboratory, Department of Electrical Engineering, State University of New York, Stony Brook, May 1992.

[8] D. Murray and B. Buxton, "Scene Segmentation From Visual Motion Using Global Optimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-9, pp. 220-228, March 1987.

[9] S. D. Roth, "Ray Casting for Modeling Solids," *Computer Graphics and Image Processing,* **18**, pp. 109-144, 1982.

[10] M. Subbarao and M.-C. Lu, "Computer Modeling and Simulation of Camera Defocus," Technical Report No. 92.01.16, Computer Vision Laboratory, Department of Electrical Engineering, State University of New York, Stony Brook, 1992. (Also appears in *Proceedings of Optics, Illumination, and Image Sensing for Machine Vision VII,* SPIE Proceedings Conf. 1822, Boston, Nov. 1992)

[11] J. Weng, T. Huang, and N. Ahuja, "Motion and Structure from Line Correspondences: Closed-Form Solution, Uniqueness, and Optimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 14, No. 3, pp. 318-336, March 1992.

Appendix A Ray Casting Algorithm

Roth [9] models solid objects by combining primitive solids, such as blocks and cylinders, using the set operators union, intersection, and difference. We have implemented his model in ANSI C and the implemented program can be easily ported to various platforms (*e.g.*, VAX, SUN, and PCs).

The program generates *3-D* scenes consisting of primitives of blocks, spheres (ellipsoids), cylinders, and cones. These primitives can be arbitrarily scaled, rotated, and translated. The input data to the program is organized as a node-based tree structure. Each node contains 14 fields:

```
struct _NODE {
    char *label;
    int address;
    char *node_type;  /*"composite" or
                        "primitive" */
    char *op;         /*"union", "intersection",
                         or "difference" */
    char *primitive;  /*"block", "sphere",
                         "cylinder" or "cone" */
    float a, b, r;    /* rotation */
    float sx, sy, sz; /* scaling */
    float tx, ty, tz; /* translation */
};
```

The leaf nodes of the tree are the primitive nodes while the internal nodes are the composite nodes. The *3-D* transform is associated with each node in the tree. This program generates *3-D* range image with hidden surface removed.

A.1 Depth Map Generation Using Ray Casting Algorithm

In this subsection, we illustrate how the object generation program can be used to generate the cone-shape depth map shown in Figure 11(b). The input description file is listed below. Note that, the first entry in this file is the number of nodes in the tree.

```
1
cone
0
primitive
union
cone
0. 0. 90.
200. 200. 200.
0. 0. 0.
```

A.2 Object Generation Using Ray Casting Algorithm

In this subsection, we illustrate how the object generation program can be used to generate a *3-D* object. This approach can be used to generate arbitrary-shape artifical objects. Figure 15 shows the *depth stop* machine part and its composite tree. The corresponding input description file is:

```
17
depth_stop
0
composite
difference
cone
30. -25. -10.
4. 4. 4.
0. 0. 160.
ds
00
```

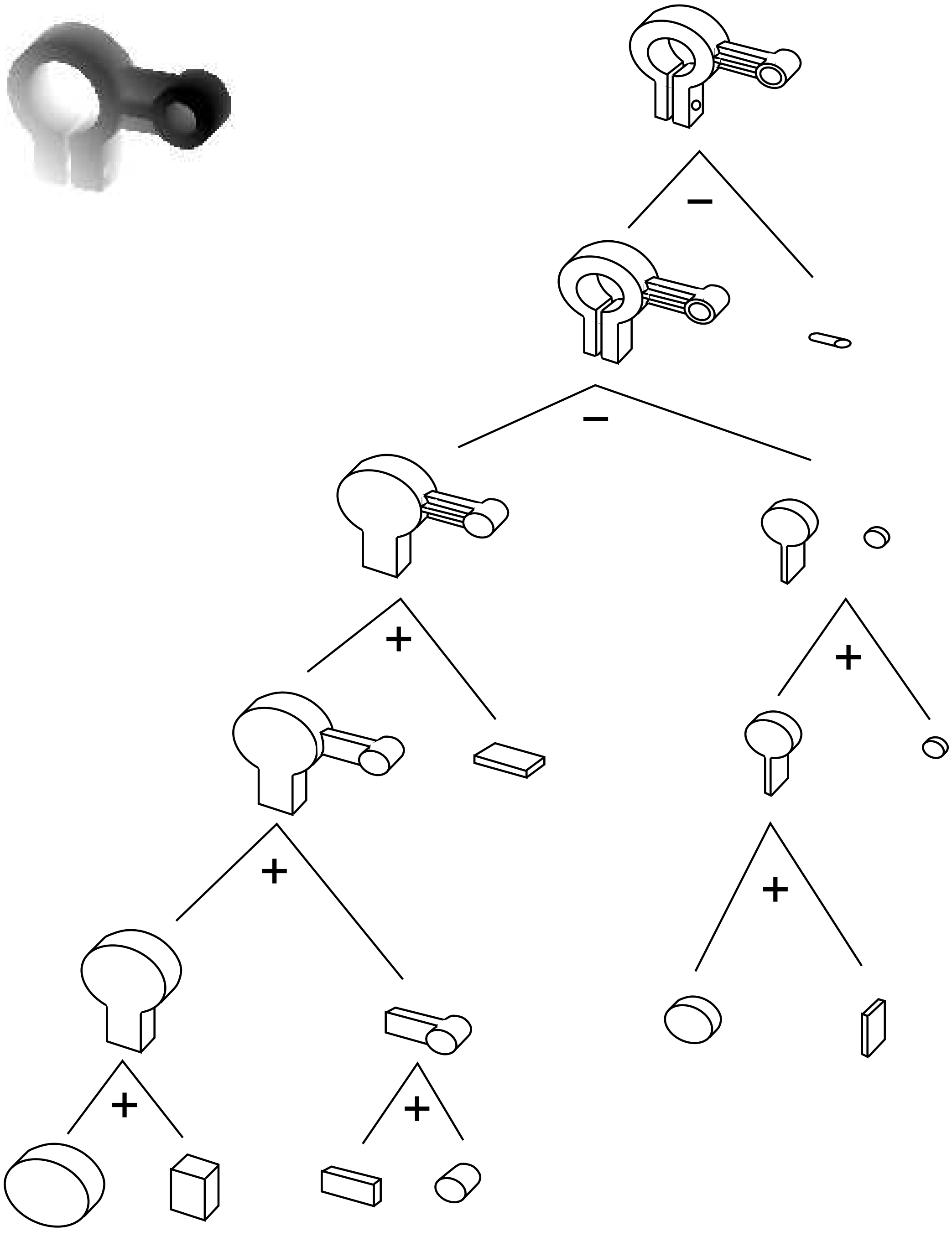


Figure 15: *Depth stop* machine part and its composite tree.

```
composite
difference
cone
0. 0. 0.
1. 1. 1.
0. 0. 0.
ds
01
primitive
union
cylinder
0. 90. 0.
4.25 4.25 40.
7. 31. -4.
ds
000
composite
union
cone
0. 0. 0.
1. 1. 1.
0. 0. 0.
ds
001
composite
union
cone
0. 0. 0.
1. 1. 1.
0. 0. 0.
ds
0000
composite
union
cone
0. 0. 0.
1. 1. 1.
0. 0. 0.
ds
0001
primitive
union
block
```

```
0. 0. 0.
35. 6. 24.
-30. -12. -12.
ds
0010
composite
union
cone
0. 0. 0.
1. 1. 1.
0. 0. 0.
ds
0011
primitive
union
cylinder
0. 0. 0.
9. 9. 28.
-39. -9. -14.
ds
00000
composite
union
cone
0. 0. 0.
1. 1. 1.
0. 0. 0.
ds
00001
composite
union
cone
0. 0. 0.
1. 1. 1.
0. 0. 0.
ds
00100
primitive
union
cylinder
0. 0. 0.
20. 20. 28.
25. -9. -14.
```

```
ds
00101
primitive
union
block
0. 0. 0.
4. 32. 28.
23. 9. -14.
ds
000000
primitive
union
cylinder
0. 0. 0.
30. 30. 24.
25. -9. -12.
ds
000001
primitive
union
block
0. 0. 0.
36. 28. 24.
7. 11. -12.
ds
000010
primitive
union
block
0. 0. 0.
64. 32. 6.
-39. -25. -3.
ds
000011
primitive
union
cylinder
0. 0. 0.
16. 16. 24.
-39. -9. -12.
```

Appendix B Syntax

B.1 Line/Batch Mode Command Syntax

The syntax of the line/batch mode commands in AVS can be expressed as the following context-free grammar where non-terminals are enclosed by left- and right-angle pairs.

```
⟨exit⟩        ::= exit | quit
⟨shell⟩       ::= shell | { ![ ⟨shell_command⟩ ] }
⟨echo⟩        ::= echo ⟨string⟩
⟨help⟩        ::= help [ object | camera | options ]
⟨read⟩        ::= read { { object ⟨filename⟩ ⟨width⟩ ⟨height⟩ } |
                         { [ parameters | depth_map ] ⟨filename⟩ } }
⟨write⟩       ::= write ⟨Filename⟩ [ from ⟨x⟩ ⟨y⟩ [ size ⟨width⟩ ⟨height⟩ ] ]
⟨view⟩        ::= view { { { object | left | right } ⟨x⟩ ⟨y⟩ } |
                         { depth_map ⟨x⟩ ⟨y⟩ } |
                         { spectrum ⟨xf⟩ ⟨yf⟩ [ normalized ] } }
⟨show⟩        ::= show { status | options | parameters }
⟨set⟩         ::= set { { { camera | object } ⟨param_id⟩=⟨param_value⟩ } |
                        { def { object | left | right } } |
                        { mode { mono | stereo } } |
                        { object ⟨x⟩ ⟨y⟩ ⟨gray_value⟩ } |
                        { option ⟨option_key⟩ } |
                        { depth_map scale ⟨real_number⟩ } }
⟨run⟩         ::= run [ spectrum { object | left | right } ]
⟨step⟩        ::= step
⟨goto⟩        ::= goto ⟨step_num⟩
⟨step_num⟩    ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
                  11 | 12 | 13 | 14 | 15 | 16
⟨x⟩           ::= 0 | ⟨positive_integer⟩
⟨y⟩           ::= 0 | ⟨positive_integer⟩
⟨xf⟩          ::= ⟨x⟩ | -⟨x⟩
⟨yf⟩          ::= ⟨y⟩ | -⟨y⟩
⟨gray_value⟩  ::= 0..255
⟨param_id⟩    ::= ⟨optical_info⟩ | ⟨obj_info⟩ | ⟨samp_info⟩ | ⟨CCD_info⟩ |
```

		⟨positionInfo⟩ | ⟨misc_info⟩
⟨optical_info⟩	::=	`f` | `s` | `f/number` | `delta_s` | `psf`
⟨obj_info⟩	::=	`u` | `Lambda` | `Vx` | `Vy` | `Vz` | `dt`
⟨samp_info⟩	::=	`Ts2` | `Ts3`
⟨CCD_info⟩	::=	`T_S` | `T_AS` | `Xs` | `Ys` | `Ts1` | `R` | `S` | `ccd_size`
⟨positionInfo⟩	::=	`Ox` | `Oy` | `Oz` | `Cx` | `Cy` | `Cz`
⟨misc_info⟩	::=	`T_LF` | `T_V` | `T_FS` | ⟨noise⟩ | ⟨transfer⟩
⟨noise⟩	::=	`N_sh` | `N_A` | `N_C` | `N_S`
⟨transfer⟩	::=	`h_sh` | `h_a` | `h_c`
⟨option_key⟩	::=	⟨border⟩ | ⟨method⟩ | ⟨io⟩
⟨border⟩	::=	`border` { ⟨mirroring⟩ | ⟨periodical⟩ | ⟨zero_pad⟩ }
⟨method⟩	::=	`method` { ⟨smart⟩ | ⟨fft⟩ | ⟨direct⟩ }
⟨io⟩	::=	`io` { ⟨ByteChar⟩ | ⟨ASCInt⟩ | ⟨BinFltMSB⟩ | ⟨BinFltLSB⟩ | ⟨ASCFloat⟩ | ⟨BinDblMSB⟩ }
⟨mirroring⟩	::=	0
⟨periodical⟩	::=	1
⟨zero_pad⟩	::=	2
⟨smart⟩	::=	0
⟨fft⟩	::=	1
⟨direct⟩	::=	2
⟨ByteChar⟩	::=	0
⟨ASCInt⟩	::=	1
⟨BinFltMSB⟩	::=	2
⟨BinFltLSB⟩	::=	3
⟨ASCFloat⟩	::=	4
⟨BinDblMSB⟩	::=	5
⟨width⟩	::=	0..1024
⟨height⟩	::=	0..1024

B.2 Parameter Syntax

The syntax of the user adjustable parameters in AVS can be expressed as the following rules.

Focal length (mm)	:	`f=`⟨number⟩
F/number	:	`f/number=`⟨number⟩
s(mm)	:	`s=`⟨number⟩
		`s=step` ⟨integer⟩
Delta s (mm)	:	`delta_s=`⟨number⟩

Wavelength Lambda (A)	:	Lambda=⟨number⟩[⟨unit⟩]
CCD pixel size (mm/pixel)	:	ccd_size=⟨number⟩
Object distance (mm)	:	u=⟨number⟩[⟨unit⟩]
Motion parameter (mm/s)	:	Vx=⟨number⟩[⟨unit⟩]
		Vy=⟨number⟩[⟨unit⟩]
		Vz=⟨number⟩[⟨unit⟩]
Camera position (mm)	:	Ox=⟨number⟩[⟨unit⟩]
		Oy=⟨number⟩[⟨unit⟩]
		Oz=⟨number⟩[⟨unit⟩]
Camera orientation (deg)	:	Cx=⟨number⟩
		Cy=⟨number⟩
		Cz=⟨number⟩
psf	:	psf=cylinder
		psf=delta(⟨i⟩,⟨j⟩)
		psf=file ⟨filename⟩ ⟨width⟩ ⟨height⟩
		psf=gaussian(⟨sigma_x⟩,⟨sigma_y⟩)
		psf=wave_optics
Light Filtering	:	T_LF(lambda)=⟨number⟩
Vignetting	:	T_V(theta,phi)=constant ⟨number⟩
		T_V(theta,phi)=gaussian(⟨sigma_x⟩, ⟨sigma_y⟩)
		T_V(theta,phi)=file ⟨filename⟩ ⟨width⟩ ⟨height⟩
Field Stop	:	T_FS(x,y)=rect(x/⟨number⟩, y/⟨number⟩)
Sensor spectral sensitivity	:	T_S(lambda)=⟨number⟩
Aperture stop	:	T_AS(t)=rect(t/⟨number⟩)
Integration over space	:	R(x,y)=rect(x/⟨number⟩, y/⟨number⟩)
		R(x,y)=file ⟨filename⟩ ⟨width⟩ ⟨height⟩
Sampling (spatial domain)	:	Xs=⟨number⟩[⟨unit⟩]
		Ys=⟨number⟩[⟨unit⟩]
Sampling (time domain)	:	Ts1=⟨number⟩[⟨unit⟩]
		Ts2=⟨number⟩[⟨unit⟩]
		Ts3=⟨number⟩[⟨unit⟩]
Sensor response	:	S(I)=I
		S(I)=I^⟨number⟩
Sample-and-hold	:	h_sh(t)=0-order
		h_sh(t)=⟨number⟩
Amplifier	:	h_a(t)=⟨number⟩
		h_a(t)=[⟨number⟩]delta(t)
Cable	:	h_c(t)=⟨number⟩
		h_c(t)=[⟨number⟩]delta(t)