6

Generating Optical Overlays

Spatial optical see-through displays overlay the real environment with computer graphics in such a way that the graphical images and the image of the real environment are visible at the same time. In contrast to head-attached or body-attached optical see-through displays, spatial displays generate images that are aligned with the physical environment. They do not follow the users' movements but rather support moving around them. They are comparable to spatial projection displays—but do not share the opaque characteristic of such displays.

An essential component of an optical see-through display is the *optical combiner*—an optical element that mixes the light emitted by the illuminated real environment with the light produced with an image source that displays the rendered graphics. Creating graphical overlays with spatial optical see-through displays is similar to rendering images for spatial projection screens for some optical combiners. For others, however, it is more complex and requires additional steps before the rendered graphics is displayed and optically combined.

While monitors, diffuse projection screens, and video projectors usually serve as light emitting image sources, two different types of optical combiners are normally used for such displays—*transparent screens* and *half-silvered mirror beam combiners*. Rendering techniques that support creating correct graphical overlays with both types of optical combiners and with different images sources will be discussed in this chapter. We make the convention that all the following elements, such as optical combiners, image sources, observers, virtual and real environments, etc. are

defined within the same *Cartesian coordinate system*. This keeps the explanations particularly simple and allows a straightforward implementation of the techniques.

6.1 Transparent Screens

Transparent screens have two main properties: they are transparent to a certain degree to allow the transmission of the image of the real environment, and they also emit the light of the rendered graphics. Observers can see directly through the screen (and through the image displayed on it) to the real environment.

In some cases, such screens are *active* and contain light-emitting elements. Examples are liquid crystal displays that are modified (by removing the opaque back light source) to enable their see-through capabilities and flexible polymers with light-emitting *transparent semi-conductors*. In other cases, external image sources, such as video or laser projectors are used to generate light that is diffused by a *transparent projection screen*. Examples include *holographic projection screens* that diffuse the light in a narrow angle to achieve an enhanced brightness for restricted viewing angles and *transparent film screens* that diffuse the light in a wide angle to support a more flexible viewing and a higher degree of transparency.

On the one hand, the use of video projectors allows building large screens that do not necessarily have to be planar. Active screens, on the

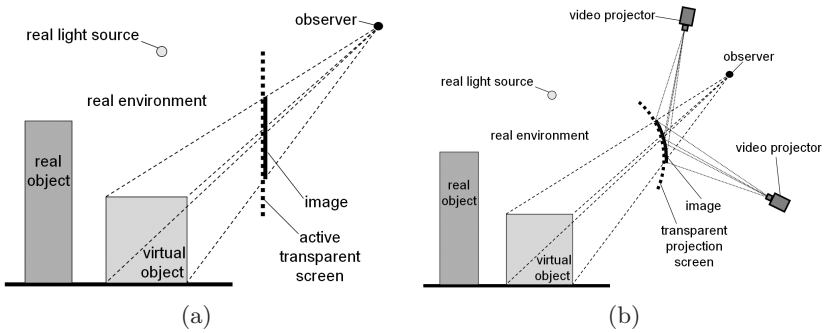

Figure 6.1. (a) Planar active transparent screen; (b) curved transparent projection screen with multiple projectors.

other hand, provide more flexibility since they are not constrained to a stationary configuration. In the future new materials, such as light emitting polymers may allow building active transparent screens that are both flexible and scalable.

Rendering for transparent screens is essentially equivalent to rendering for regular projection screens or monitors. While for planar screens an affine off-axis projection transformation is sufficient, *curvilinear image warping* is required for curved screens. For large or extremely curved screens, the image has to be composed of the contribution displayed by multiple projectors. The different pieces have to be geometrically aligned and blended to result in a consistent final image. All these techniques are explained in detail for opaque screens in Chapters 3 and 5. They are the same for transparent screens.

Sections 6.3 through 6.5 describe a rendering framework for spatial optical see-through displays that use mirror beam combiners. Most of this framework, such as refraction correction and multi-screen and multi-plane beam combiner transformations, is exactly the same for transparent screens. The main difference to the mirror beam combiner is that reflection transformations do not apply in this case. They have to be ignored if the framework is used for displays with transparent screens. If projection displays are used to generate images, the outlined screen transformation is equivalent to the rendering techniques explained in Chapters 3 and 5.

An important fact that needs to be mentioned is that the rendered image appears directly on the surface of the transparent screen. Similar to head-attached displays, this causes focus problems if the real environment to be augmented is not located in the same place as the image. For transparent screens, this can never be the case since the screen itself (and therefore the image) can never take up exactly the same space within the environment. It is not possible for our eyes to focus on multiple distances simultaneously. Thus, in extreme cases, we can only continuously shift focus between the image and the real environment, or perceive either one unfocused.

If stereoscopic graphics need to be displayed on transparent projection screens, we must pay attention to the materials used. Not all materials preserve the polarization of light that is produced by *passive stereoscopic projection* setups—especially when materials are bent to form curved screens. Transparent film screens, for instance, have the property to preserve polarization in any case—even if bent. Active stereo projection systems do not cause such problems.

6.2 Mirror Beam Combiners

Mirror beam combiners are the most common optical combiners because of their availability and low cost. If they cannot be obtained from an optics store, they can be homemade in almost any size and shape. For instance, regular float glass or Plexiglas can be coated with a *half-silvered film*, such as 3M's Scotchtint sun protection film. These materials are easily available in hardware stores. The advantage of half-silvered film is that it can be coated onto a flexible carrier, such as thin Plexiglas, that can easily be bent to build curved displays. The drawback of such a material is its non-optimal optical properties. Instead of having transmission and reflection factors of 50%, these materials normally provide factors of approximately 70% reflection and 30% transmission, due to their sun-blocking functionality. An alternative material is the so-called *spyglass* that offers better and varying transmission/reflection factors without sun-blocking layers. However, the reflective film is usually impregnated into float glass—thus it is not well suited for building curved mirror displays.

Spatial optical see-through displays that apply mirror beam combiners as optical combiners have to display the rendered graphics on a *secondary screen* that is reflected by the mirror optics. The observer sees through the optical combiner and through the reflected image at the real environment. The image that appears on the secondary screen usually should not be visible. To hide it, the secondary screen is often coated with *light directing film*, such as 3M's Light Control Film. Such a film can be used to direct the displayed image only towards the mirror optics and not to the observer. Thus, only the reflection of the displayed image can be seen and not the image itself.

In contrast to transparent screens, mirror beam combiners create an optical reflection of the secondary screen and the displayed image. There are no physical constraints in aligning the image closer to the real environment. The reflected image can even intersect with the real environment. Consequently, the focus problem of transparent screens can be improved although not completely solved.

6.3 Planar Mirror Beam Combiners

To render a three-dimensional graphical scene on spatial optical see-through displays applying planar mirror beam combiners requires neutralizing the optical transformations that are caused by half-silvered mirrors: reflection

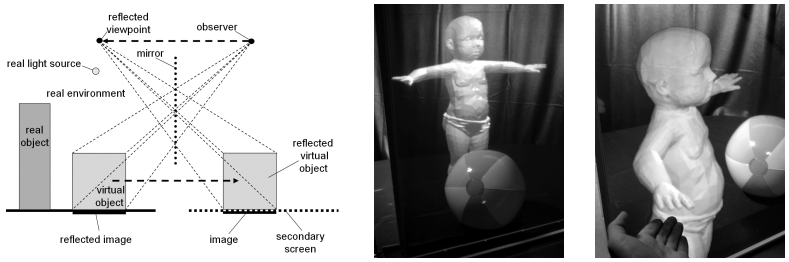

Figure 6.2. (a) Affine reflection transformation for planar mirror beam combiner; (b) a virtual baby observed through a large beam combiner reflecting a horizontal projection screen; (c) due to parallax effects, virtual objects (e.g., the baby's arm) can appear in front of the mirror. (*Images (b) and (c) reprinted from [10]* © *MIT Press.*)

and refraction. The goal is to render the graphical content and display it on the secondary screen in such a way that its reflection appears perspectively correct and aligned with the real environment.

6.3.1 Reflection

A planar mirror beam combiner divides the environment into two subspaces: the one that contains the observer and the secondary screen, and the one that contains the real environment to be augmented and the physical light sources that illuminate it.

Note that from a geometric optics point of view, the real environment behind the mirror equals the mirror's *image space* (i.e., the reflection that appears behind the mirror by looking at it).

Virtual objects that consist of graphical elements (such as geometry, normal vectors, textures, clipping planes, virtual light sources, etc.) are defined within the same global world coordinate system in which the real environment, the observer, the secondary screen, and the mirror beam combiner are located. In contrast to conventional virtual reality scenarios, this coordinate system actually exceeds the boundaries of the display screen and extends into the surrounding real environment (see Figure 6.2).

The virtual objects are either defined directly within the real environment or they are transformed to it during an *object registration process*.

We now consider the mirror and compute the reflection of the observer's physical eye locations (as well as possible virtual headlights). We then apply the inverse reflection to every graphical element that is located within

the real environment. In this manner these graphical elements are transformed and can be projected and displayed on the secondary screen. The displayed image is optically reflected back by the mirror into the real environment (or optically, into the mirror's image space).

When the setup is sufficiently calibrated, the real environment and the mirror's image space overlay exactly. The virtual objects appear in the same position within the image space as they would within the real environment without the mirror (if a direct display possibility was given within the real environment).

Note that the *reflection transformation* of planar mirrors is a *rigid-body transformation*, and preserves all properties of the transformed geometry.

As discussed in Chapter 2, with known plane parameters of the mirror beam combiner within the world coordinate system, a point in three-dimensional space can be reflected by

$$p' = p - 2(np + d)n, \tag{6.1}$$

where p' is the reflection of p over the mirror plane $[n, d] = [a, b, c, d]$.

This is equivalent to multiplying p with the 4×4 *reflection matrix*

$$R = \begin{bmatrix} 1-2a^2 & -2ab & -2ac & -2ad \\ -2ab & 1-2b^2 & -2bc & -2bd \\ -2ac & -2bc & 1-2c^2 & -2cd \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that $R = R^{-1}$.

The reflected viewpoint e' of the observer (which, for instance, is head-tracked) can be computed using Equation (6.1) or by multiplying the original viewpoint e with the reflection matrix.

The inverse reflection of the virtual scene that is located within the real environment is simply computed from its reflection with respect to the mirror plane. Since we assume that the real environment and the mirror's image space exactly overlay, we can also assume that the reflection of the graphical elements located within the real environment results in the inverse reflection of the image space; that is, they are transformed to their corresponding positions on the observer's side of the mirror and can be displayed on the secondary screen. Consequently, the additional model transformation (i.e., the inverse reflection of the scene) is achieved by multiplying the reflection matrix onto the current model-view matrix of the transformation pipeline (between scene transformation and view transformation).

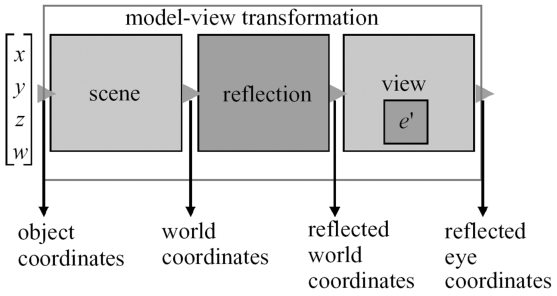

Figure 6.3. The integration of reflection transformations into the rendering pipeline.

Consequently, geometry that is registered to the real environment is first transformed from object coordinates into the coordinates of the world coordinate system, then into *reflected world coordinates*, and finally into *reflected eye coordinates*. After the model-view transformation has been applied, the rendering pipeline is continued in the normal way (see Chapter 2); the reflected eye coordinates are off-axis projected into clip coordinates, then, after the perspective division, transformed into normalized device coordinates, and finally (via the viewport transformation) mapped into window coordinates.

By applying the reflection matrix, every graphical element is reflected with respect to the mirror plane. A side effect of this is that the order of reflected polygons is also reversed (e.g., from counterclockwise to clockwise) which, due to the wrong front-face determination, results in a wrong rendering (e.g., lighting, culling, etc.). This can easily be solved by explicitly reversing the *polygon order*. Note that transformations and rendering have to be done for both viewpoints (left and right) if stereoscopic rendering is activated.

The reflection transformation can be entirely executed by accelerated graphics hardware that provides a fixed function rendering pipeline. The following OpenGL code fragment can be used to configure the rendering pipeline with reflection transformation:

```
...

// mirror plane = a,b,c,d
// viewpoint = e[0..2]
// world coordinate system = x/y-axes horizontal plane,
```

```
// z-axis points up
float e[3], e_[3];
float NP;
float R[16];

// compute reflected viewpoint e_ from original viewpoint e
NP = a * e[0] + b * e[1] + c * e[2];
e_[0]= e[0] - 2.0 * (NP + d) * a;
e_[1]= e[1] - 2.0 * (NP + d) * b;
e_[2]= e[2] - 2.0 * (NP + d) * c;

// set up reflection matrix
R[0]  = 1-2*a*a; R[1]  =-2*a*b;   R[2]  =-2*a*c;   R[3] =0;
R[4]  =-2*a*b;   R[5]  = 1-2*b*b; R[6]  =-2*b*c;   R[7] =0;
R[8]  =-2*a*c;   R[9]  =-2*b*c;   R[10] = 1-2*c*c; R[11]=0;
R[12] = -2*a*d;  R[13] =-2*b*d;   R[14] =-2*c*d;   R[15]=1;

// configure rendering pipeline
glViewport(...);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(...);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(e_[0], e_[1], e_[2], e_[0], e_[1], 0, 0, 1, 0);
glMultMatrixf(R);

// reverse polygon order
glFrontFace(GL_CW + GL_CCW - glGetIntegerv(GL_FRONT_FACE));

// draw scene with scene transformation

...
```

An alternative to a reflection of the entire geometry and viewpoint is to set up a viewing frustum that is defined by the reflected image plane instead of the image plane on the physical secondary screen. In Figure 6.2(a), the secondary screen is reflected from the right sides of the beam combiner to its left side. The unreflected viewing frustum (right side) can be used if defined relative to the reflected image plane (left side). For this, the exact position and orientation of the reflected image plane has to be known. They can also be derived from the parameters of the mirror plane.

In addition, the displayed image has to be reflected over the on-screen axis that is perpendicular to the two-dimensional intersection vector (i_x, i_y) of the mirror plane and the secondary screen. In a simple example, this in-

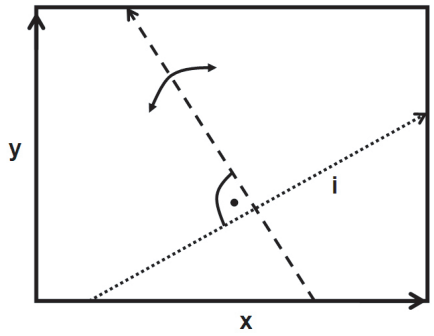

Figure 6.4. Simple example for reflection over on-screen axis.

tersection equals the x-axis (or the y-axis) of the screen coordinate system, as illustrated in Figure 6.4. In this case, an additional scaling transformation can be applied after the projection transformation that causes the fragments to be reflected within the normalized device space. For instance, `glScale(i_x, i_y, 0)` with $i_x = -1, i_y = 0$ causes a reflection over the y-axis for the case that the intersection of the mirror with the screen is on the x-axis (all in normalized screen/device coordinates). For an arbitrary intersection vector, however, a scaling transformation alone is not sufficient. An additional rotation transformation is required that first aligns the intersection vector with either the x- or the y-axis in the screen coordinate system. Then the scaling is transformed along this principle axis (for example over the x-axis). Finally the reverse rotation transformation has to be applied to produce the correct effect.

It is important to apply this reflection transformation after the projection (e.g., before `glFrustum()` in OpenGL's reversed matrix stack notation) since it has to be the final transformation, and it must not influence other computations, such as lighting and depth culling.

An interesting optical effect can be observed by applying mirrors in combination with stereoscopic secondary screens; convex or planar mirrors can optically only generate virtual images. However, in combination with stereoscopic graphics and the effects caused by stereopsis, virtual objects can appear in front of the mirror optics (Figure 6.2(c)). We refer to this effect as *pseudo real images*. In nature, real images of reflected real objects can only be generated with concave mirrors. Note that a restricted direct manipulative interaction with pseudo real images in front of the mirror optics is supported.

6.3.2 Refraction

From an optics point of view, the glass or Plexiglas carriers used for optical combiners (i.e., mirror beam combiner or transparent screens) are lenses that cause refraction distortion. A homogeneous medium that is bound by two plane-parallel panels is referred to as a planar lens. The refraction distortion is small and can be neglected for thin planar lenses, but has to be corrected for thick plates. Note that the following techniques are also applicable for transparent screens that suffer from refraction distortion.

All virtual objects that are registered to the real environment are virtual points that are not physically located behind the optical combiner. They are images that are created by the optical combiner. Mirror beam combiners are usually *front surface mirrors* (i.e., the mirror film coated on the side of the carrier that faces the image source and the observer) while transparent screens can be front projected causing the same registration problem: the displayed image is not physically refracted by the optical combiner. However, the transmitted light which is emitted by the real environment and perceived by the observer, is refracted. Consequently, the transmitted image of the real environment cannot be precisely registered to the reflected virtual objects, even if their geometry and alignment match exactly within our world coordinate system. Unfortunately refraction cannot be undistorted by a rigid-body transformation, but approximations exist that are sufficient for augmented reality display types.

All optical systems that use any kind of see-through element have to deal with similar problems. For head-mounted displays, aberrations (optical distortion) caused by refraction of the integrated lenses are mostly assumed to be static [4]. Due to the lack of eye-tracking technology as a component of head-mounted displays, the rotation and the exact position of the eyeballs as well as the movement of the optics in front of the observer's eyes is not taken into account. Thus, a static refraction distortion is precomputed for a *centered on-axis* optical constellation with methods of *paraxial analysis*. The result is stored in a *two-dimensional look-up table*. During rendering, this look-up table is referenced to transform rendered vertices on the image plane before they are displayed. Rolland and Hopkins [162], for instance, describe a polygon-warping technique that uses the look-up-table to map projected vertices of the virtual objects' polygons to their predistorted location on the image plane. This approach requires subdividing polygons that cover large areas on the image plane. Instead of predistorting the polygons of projected virtual objects, the projected

image itself can be predistorted, as described by Watson and Hodges [200], to achieve a higher rendering performance.

For spatial see-through displays, however, aberrations caused by refraction are dynamic, since the optical distortion changes with a moving viewpoint that is normally off-axis and off-centered with respect to the optical combiner.

For some *near-field displays*, such as reach-in displays, the displacement caused by refraction can be estimated [204]. An estimation of a constant refraction might be sufficient for near-field displays with a fixed viewpoint that use a relatively thin beam combiner. For larger spatial optical see-through setups that consider a head-tracked observer and apply a relatively thick beam combiner, more precise methods are required. In the following explanation, we want to consider such a display constellation.

Since we cannot predistort the refracted transmitted image of the real environment, we artificially refract the virtual scene before it is displayed, in order to make both images match. In contrast to the static transformation that is assumed for head-mounted displays, refraction is dynamic for spatial optical see-through displays and cannot be precomputed. In general, refraction is a complex curvilinear transformation and does not yield a *stigmatic mapping* (the intersection of multiple light rays in a single image point) in any case; we can only approximate it.

In the case of planar lenses, light rays are refracted twice—at their entrance points and at their exit points. This is referred to as *in-out refraction*. In the case of planar lenses, the resulting out-refracted light rays have the same direction as the corresponding original rays, but they are shifted by the amount Δ parallel to their original counterparts. Due to refraction, an object that is physically located at the position p_o appears at the position p_i. To maintain registration between a virtual object (that will appear at position p_o) and the corresponding real object (that will appear at position p_i), the virtual object has to be repositioned to p_i.

The offset Δ can be computed as

$$\Delta = t\left(1 - \frac{\tan\alpha_t}{\tan\alpha_i}\right),$$

where t is the thickness of the planar lens, α_i is the angle between the plane normal of the plate and the line of sight, and α_t is given by *Snell's law of refraction*:

$$\eta_1 \sin\alpha_i = \eta_2 \sin\alpha_t.$$

It is constrained to the following boundaries:

$$\lim\left(\alpha_i \to \frac{\pi}{2}\right) \Rightarrow \Delta = t,$$
$$\lim\left(\alpha_i \to 0\right) \Rightarrow \Delta = t\left(1 - \frac{\sin\alpha_t}{\sin\alpha_i}\right) = t\left(1 - \frac{1}{\eta_2}\right) = \text{const.}$$

A special case of the previous transformation is an on-axis situation where the incidence angle is perpendicular to the lens and parallel to the optical axis (i.e., $\alpha_i = 0$). In this situation, the offset equation can be simplified to $\Delta = t\,(1 - 1/\eta_2)$.

In the case of an optical combiner that is located in air, the refraction index η_1 is equal to 1. The refraction index η_2 is that of the carrier's base material (e.g., glass or Plexiglas).

A simple solution to simulate refraction is the assumption that, despite its optical nature, it can be expressed as a rigid-body transformation. This is a common approximation used by the three-dimensional computer graphics community to render realistic-looking refraction effects in real time. In beam-tracing [61], for instance, it was assumed that, considering only *paraxial rays* (entering light rays that are exactly or nearly perpendicular to the refracting plane), objects seen through a polygon with a refraction index of η appear to be η times their actual distance. This is because light travels slower in denser materials by precisely this factor. For this approximation, the incidence angles of the optical line of sight are not taken into account but, instead, a constant incidence angle of $\alpha_i = 0$ to the optical axis is assumed. This, however, covers on-axis situations only. For off-axis situations that occur with spatial optical see-through displays, the incident

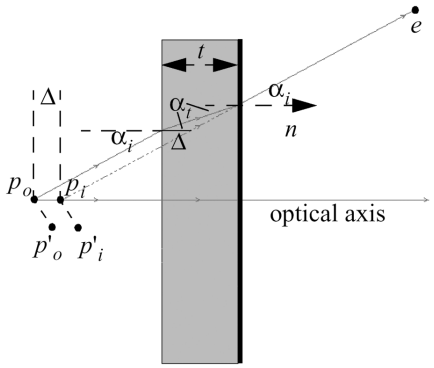

Figure 6.5. Off-axis refraction transformation for planar lenses.

angle has to be considered. As a rigid-body transformation, refraction can be expressed by a homogeneous 4×4 matrix:

$$F = \begin{bmatrix} 1 & 0 & 0 & \Delta a \\ 0 & 1 & 0 & \Delta b \\ 0 & 0 & 1 & \Delta c \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The advantage of this simple translation along the plate's optical axis is that it can be integrated into a fixed function rendering pipeline and can be carried out by graphics hardware. The drawback, however, is that this is only a rough approximation for refraction distortion, since every vertex is translated by the same amount Δ which is computed from a common incident angle (such as the angle between the viewing direction and the optical axis, for instance). The curvilinear characteristics of optical refraction and the individual incident angles of the viewing vectors to each scene vertex are not taken into account.

Programmable rendering pipelines allow per-vertex transformations directly on the graphics hardware. Thus the correct *curvilinear refraction transformation* can be implemented as a vertex shader, rather than as a rigid-body transformation expressed by a homogeneous matrix.

The following Cg shader fragment can be used to configure a programmable rendering pipeline with the refraction transformation:

```
...

// viewpoint = e[0..2]
// vertex = v[0..3]
// plane normal = n[0..2]
// plane thickness = t
// refraction index of material = r
float3 n,d;
float alpha_i, alpha_t, delta;

// compute viewing vector to vertex
d = e - v;
d = normalize(d);
n = normalize(n);

// compute angle between normal and viewing vector
alpha_i = acos(dot(d,n));

// compute delta
alpha_t = asin(sin(alpha_i) / r);
```

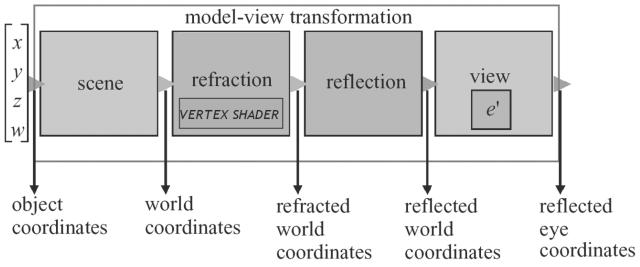

Figure 6.6. The integration of reflection transformations into the rendering pipeline.

```
delta = t * (1 - (tan(alpha_t)) / tan(alpha_i));

// compute refracted vertex
v.xyz = v.xyz + n * delta;

...
```

Whether refraction is implemented as a rigid-body transformation or as a per-vertex transformation, the sequence of the transformations carried out by the rendering pipeline is important for spatial optical see-through displays.

Figure 6.6 illustrates the extension of Figure 6.3. The refraction transformation has to be carried out after the scene transformation (either a rigid-body or a per-vertex transformation). Vertices are transformed from world coordinates to *refracted world coordinates* first. If the optical combiner is a mirror beam combiner, the refracted vertices are then reflected. If the optical combiner is a transparent screen, reflection transformation is not used. Finally, the vertices are mapped into reflected eye coordinates (either with the reflected or with the unreflected viewpoint, depending on the optical combiner), projected, converted into window coordinates, rasterized, and displayed.

The pseudocode below summarizes the rendering steps for spatial optical see-through displays that use one planar screen and one planar beam combiner. Only one user is supported.

```
for left and right viewpoints i
  initialize transformation pipeline and polygon order
  compute reflected viewpoint e'_i = Re_i
  compute refraction offset Δ_i for i
  set transformation pipeline: RMF_iV_iP
```

```
  reverse polygon order
  render scene from e'_i
endfor
```

First, the polygon order and the transformation pipeline have to be set to an initial state. Then the reflected viewpoint and the view-dependent refraction offset (Δ_i) are computed. The transformation pipeline is then set to the corresponding concatenation of transformation matrices: reflection transformation (R), model transformation (M), refraction transformation (F_i, with e_i), view transformation (V_i, with e'_i), and the projection transformation (P). Finally, the polygon order is reversed and the scene is rendered. Note that we assume that the screen coordinate system is equivalent to the world coordinate system. Note also that R might not be static but has to be recomputed continuously (e.g., if moving components have to be supported—see Section 6.5).

6.4 Screen Transformation and Curved Screens

If the coordinate system of the secondary screen does not match with the world coordinate system, an additional *screen transformation* has to be added to the rendering pipeline. It expresses the screen's transformation within the world coordinate system and is usually a composition of multiple translation and rotation transformations. Similar to the reflection transformation, not only the reflected geometry but also the reflected viewpoint has to be mapped into the *screen coordinate system*.

Once again the sequence in which all transformations are carried out is important. The screen transformation is applied after the reflection trans-

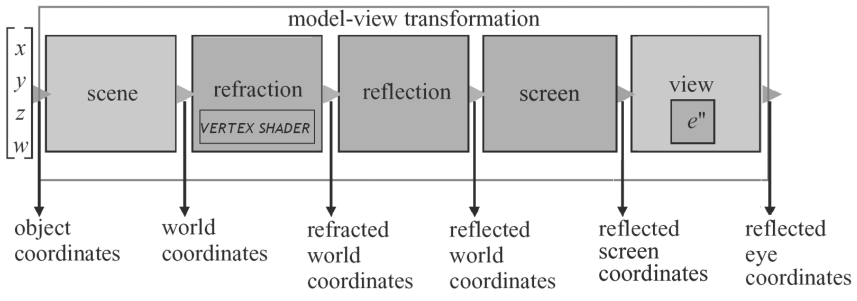

Figure 6.7. The integration of screen transformations into the rendering pipeline.