Development of Games

Lecture 5

Rendering, lighting, camera

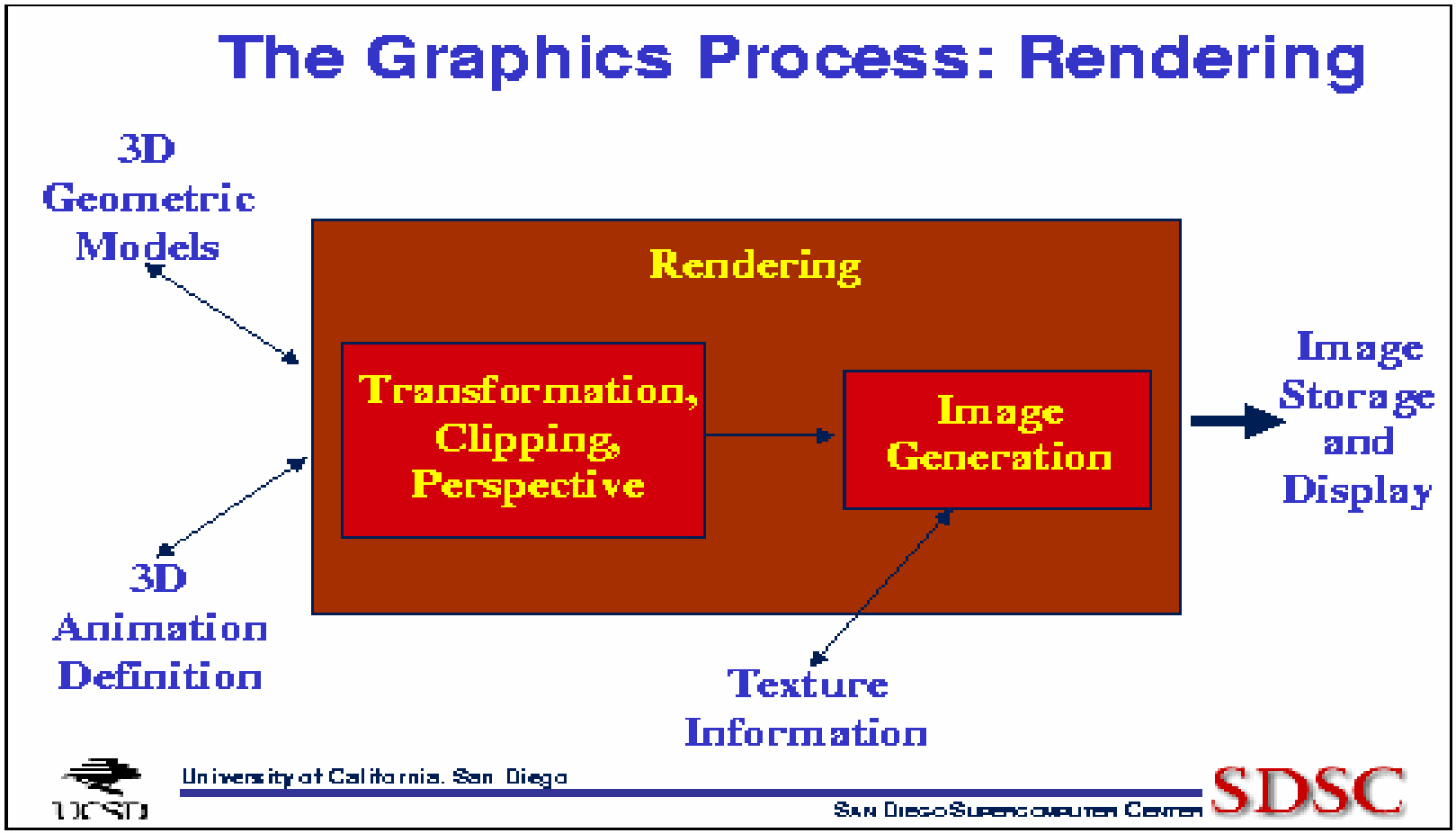
Rendering

- **Render**

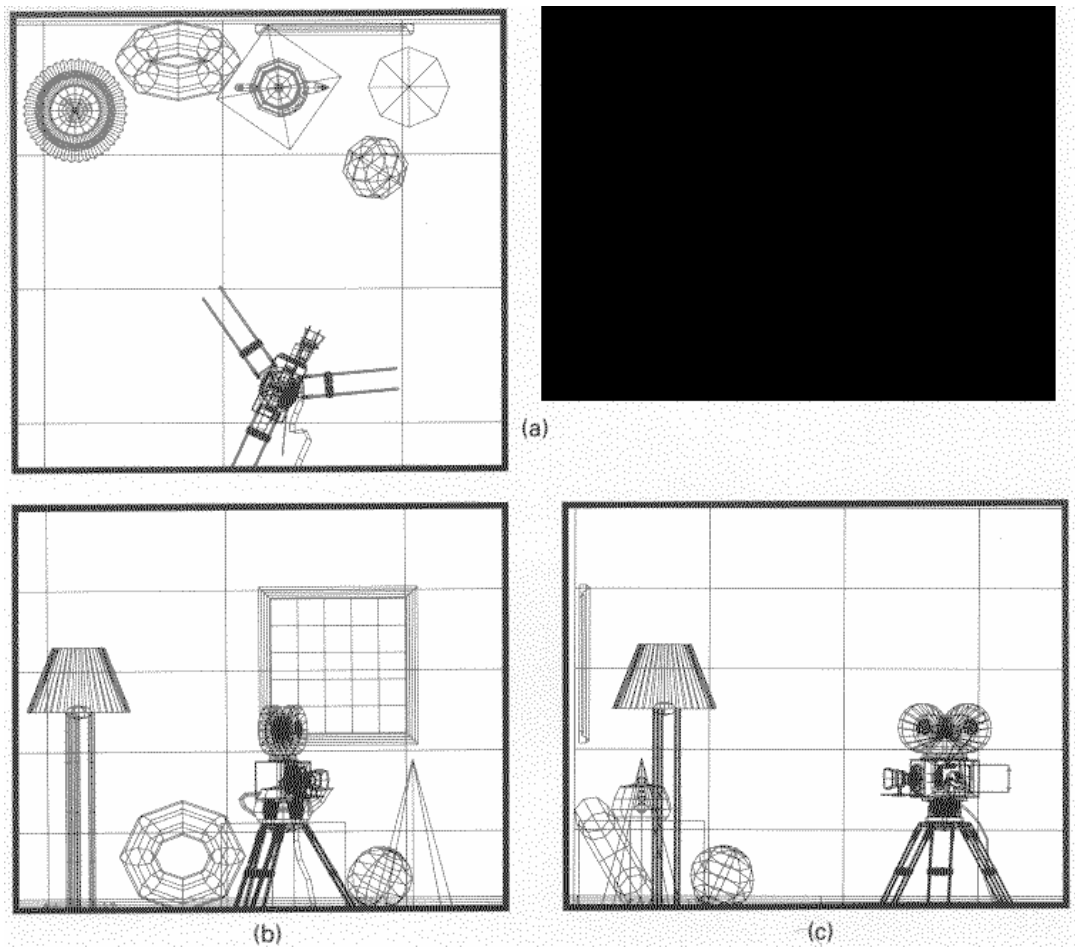
  1: To compute an entier scene (as an output array of pixels) from a graphics data base.

  2: To convert a graphics primitive into individual pixels

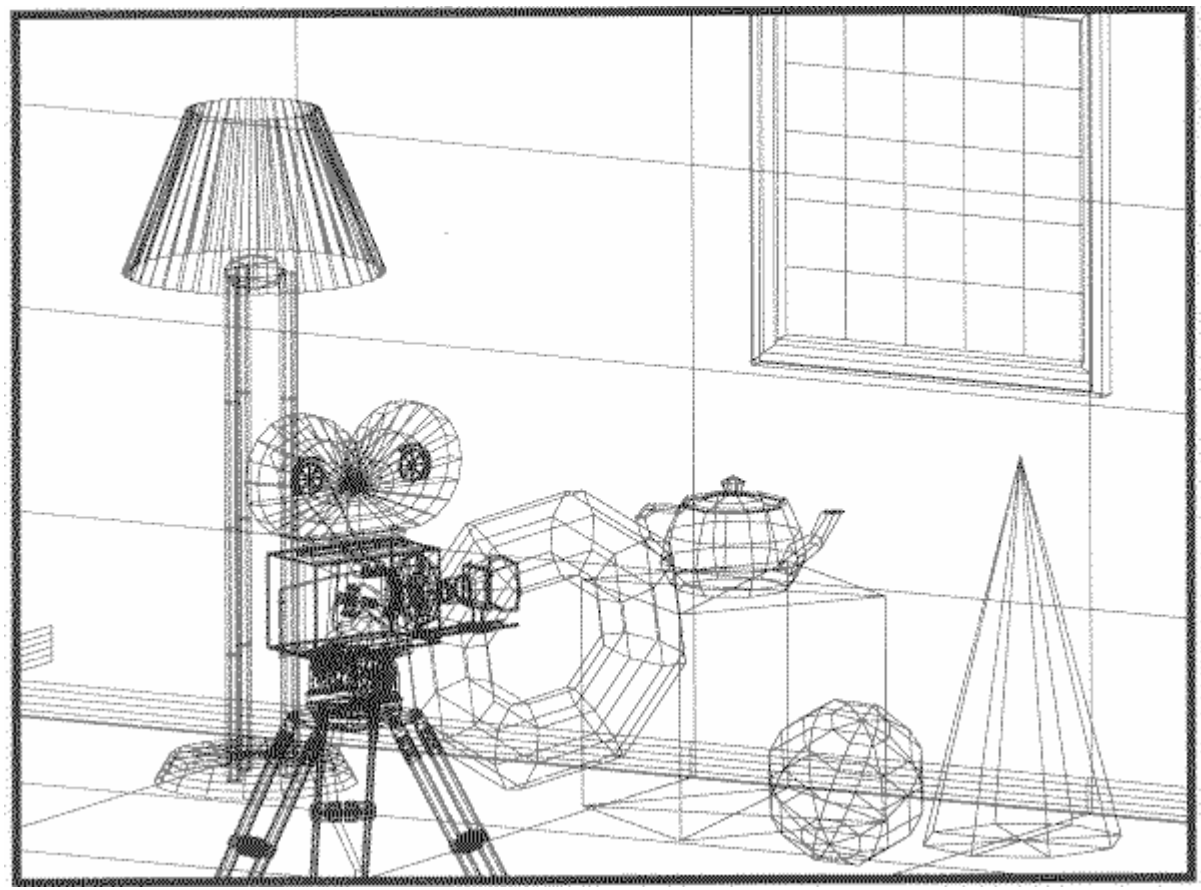
Rendering (2)


The Graphics Process: Rendering
3D Geometric Models
Rendering
Transformation, Clipping, Perspective
Image Generation
Image Storage and Display
3D Animation Definition
Texture Information
University of California, San Diego
SDSC
San Diego Supercomputer Center

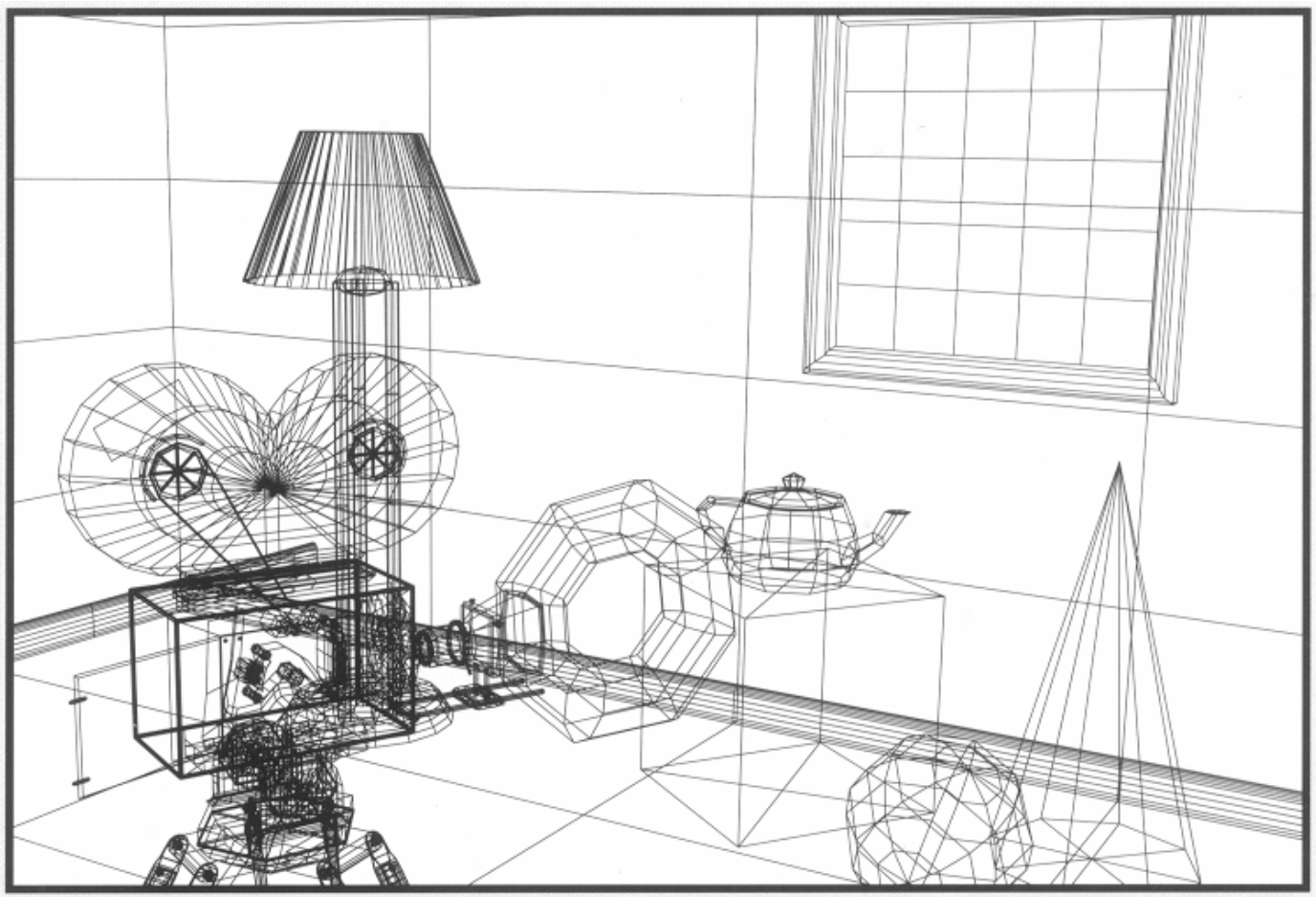
Orthographic parallel projection



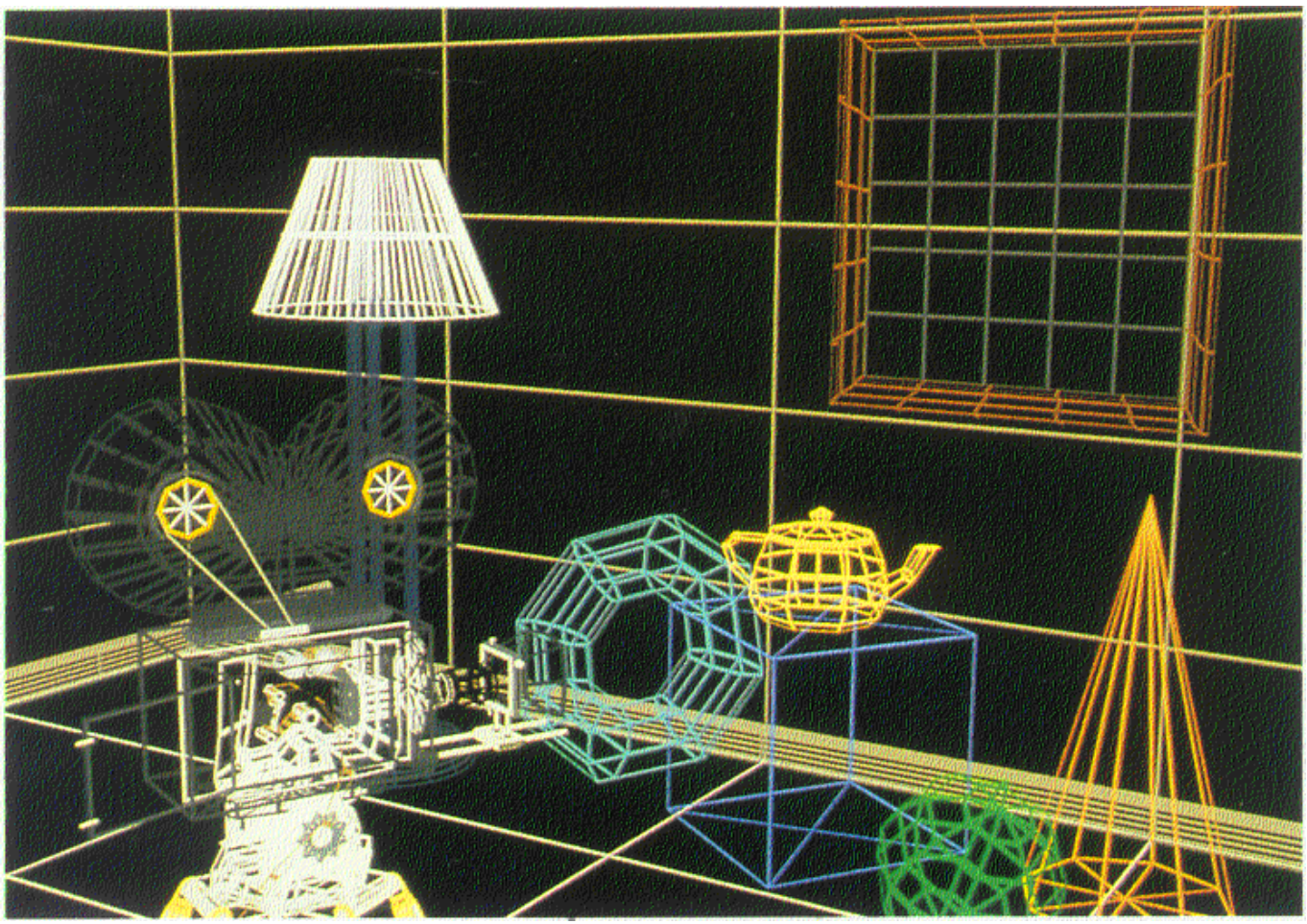
Oblique parallel projection



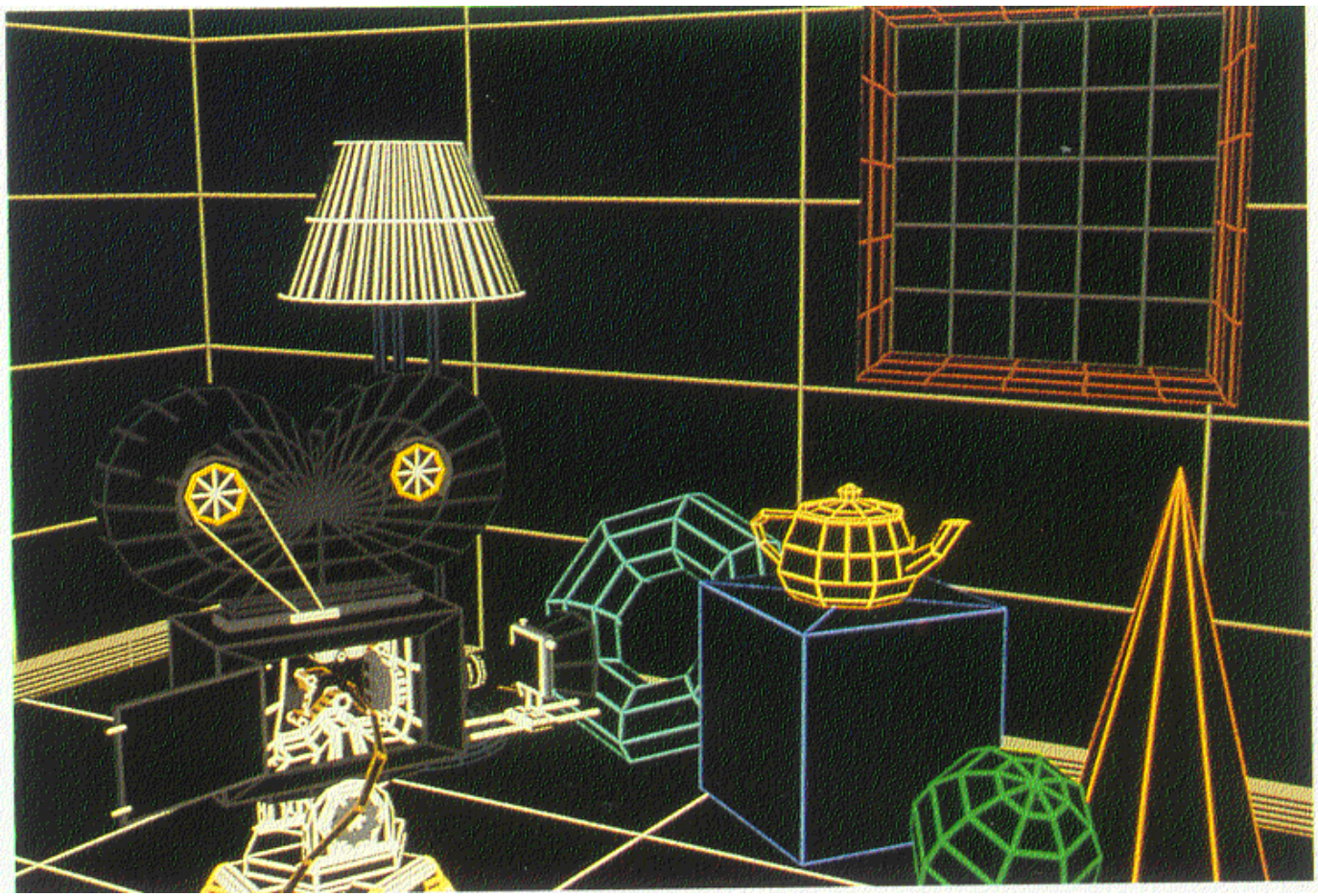
Perspective projection



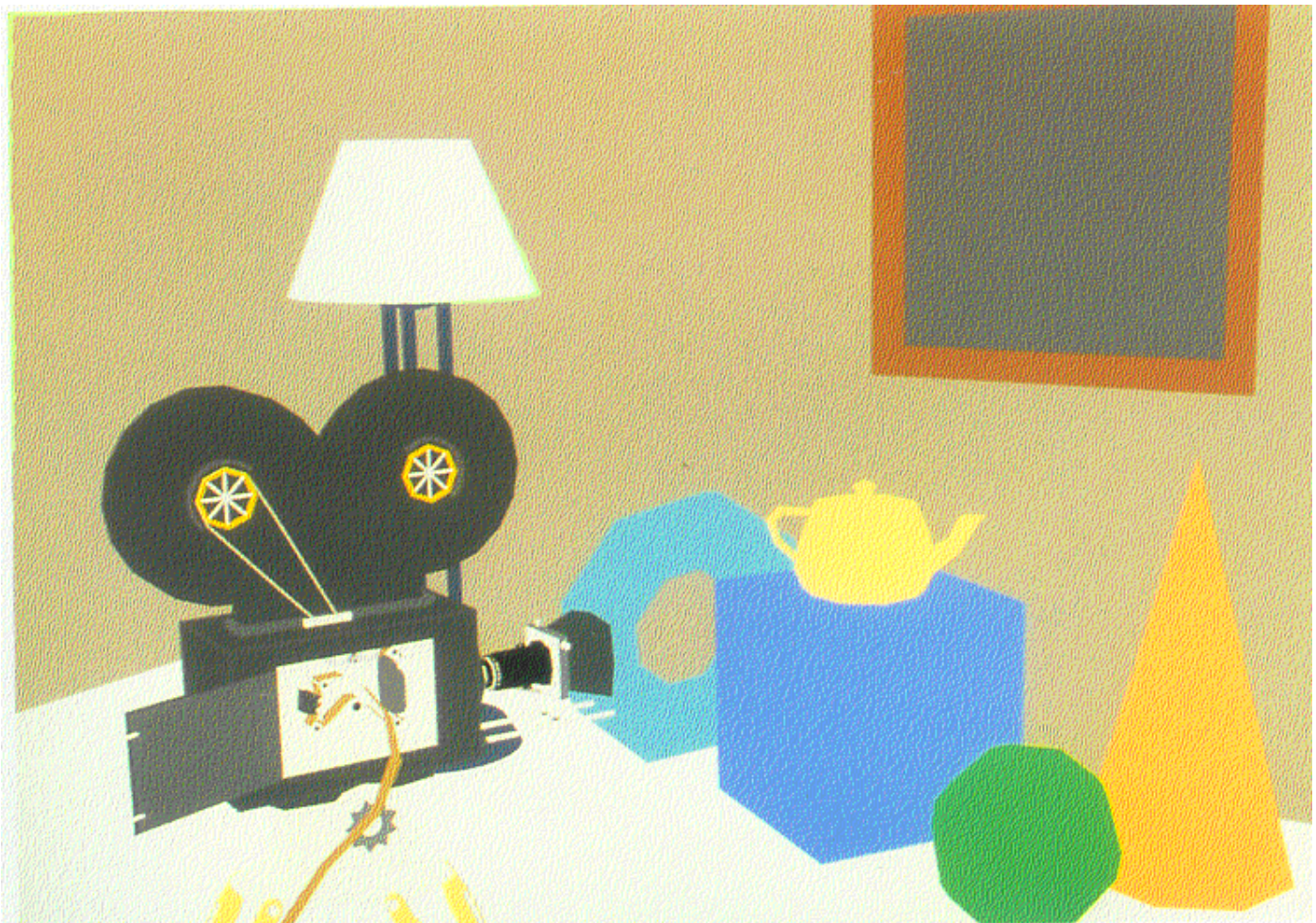
Color lines



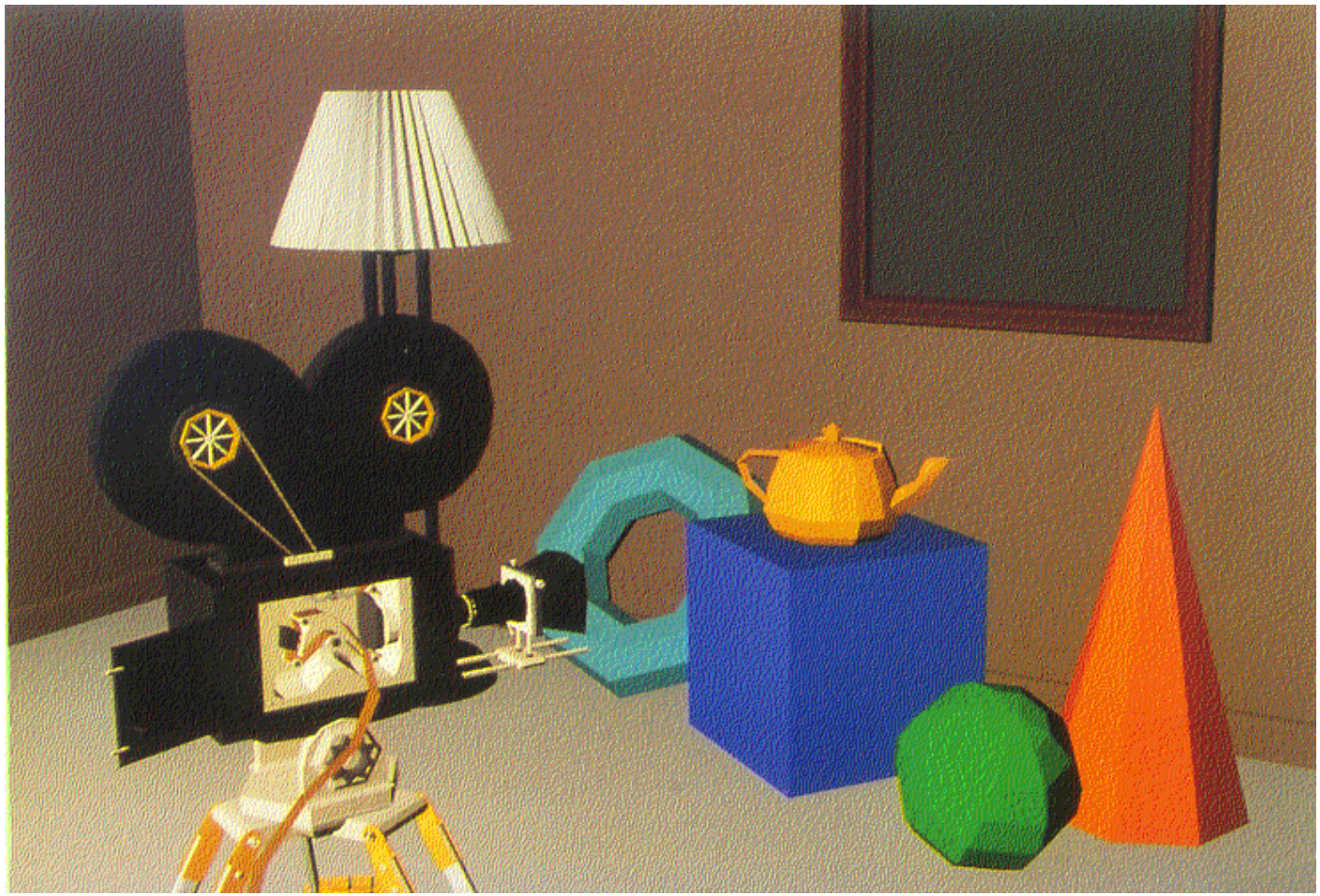
Removing of invisible lines



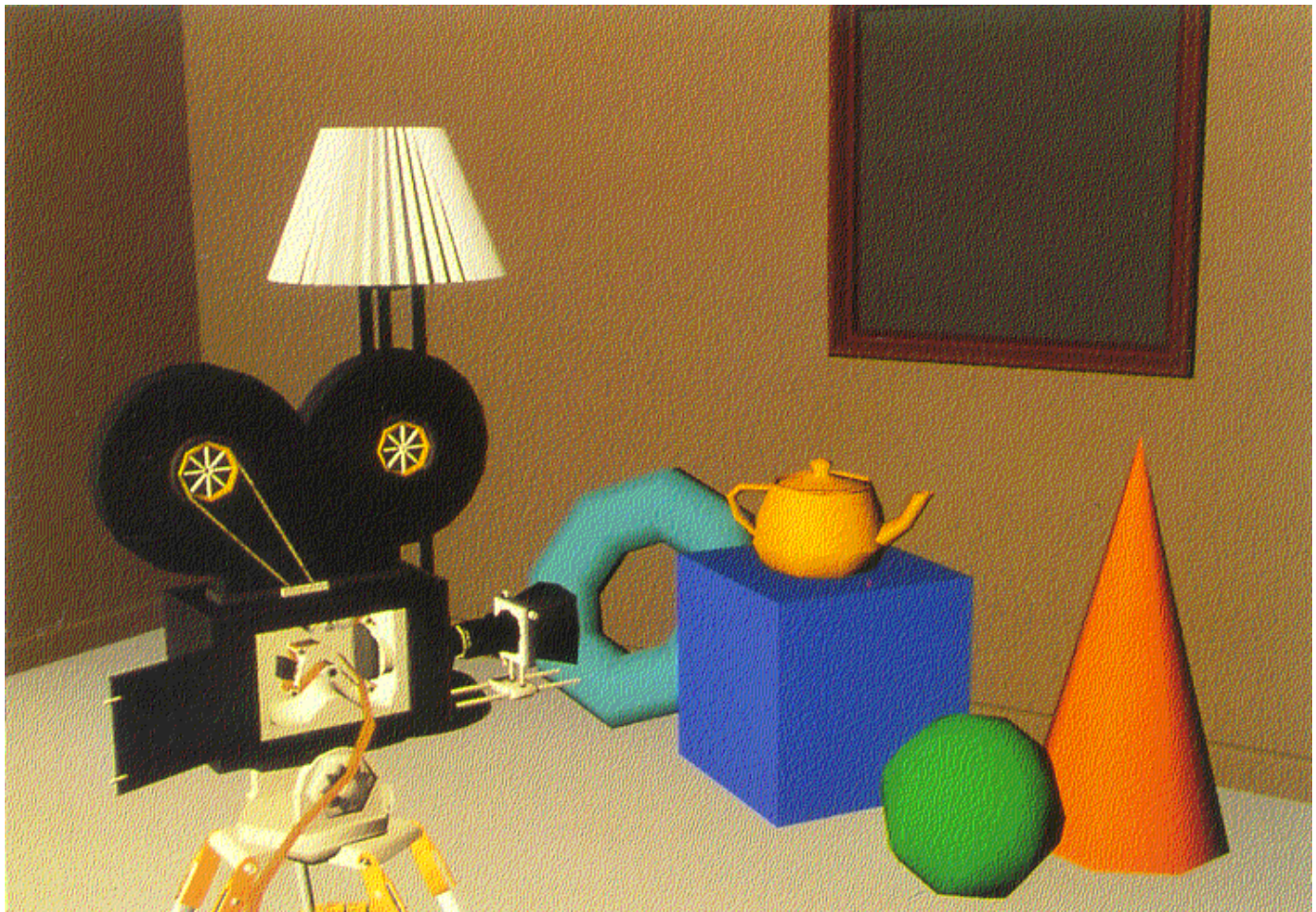
Determination of visible surfaces (ambient light)



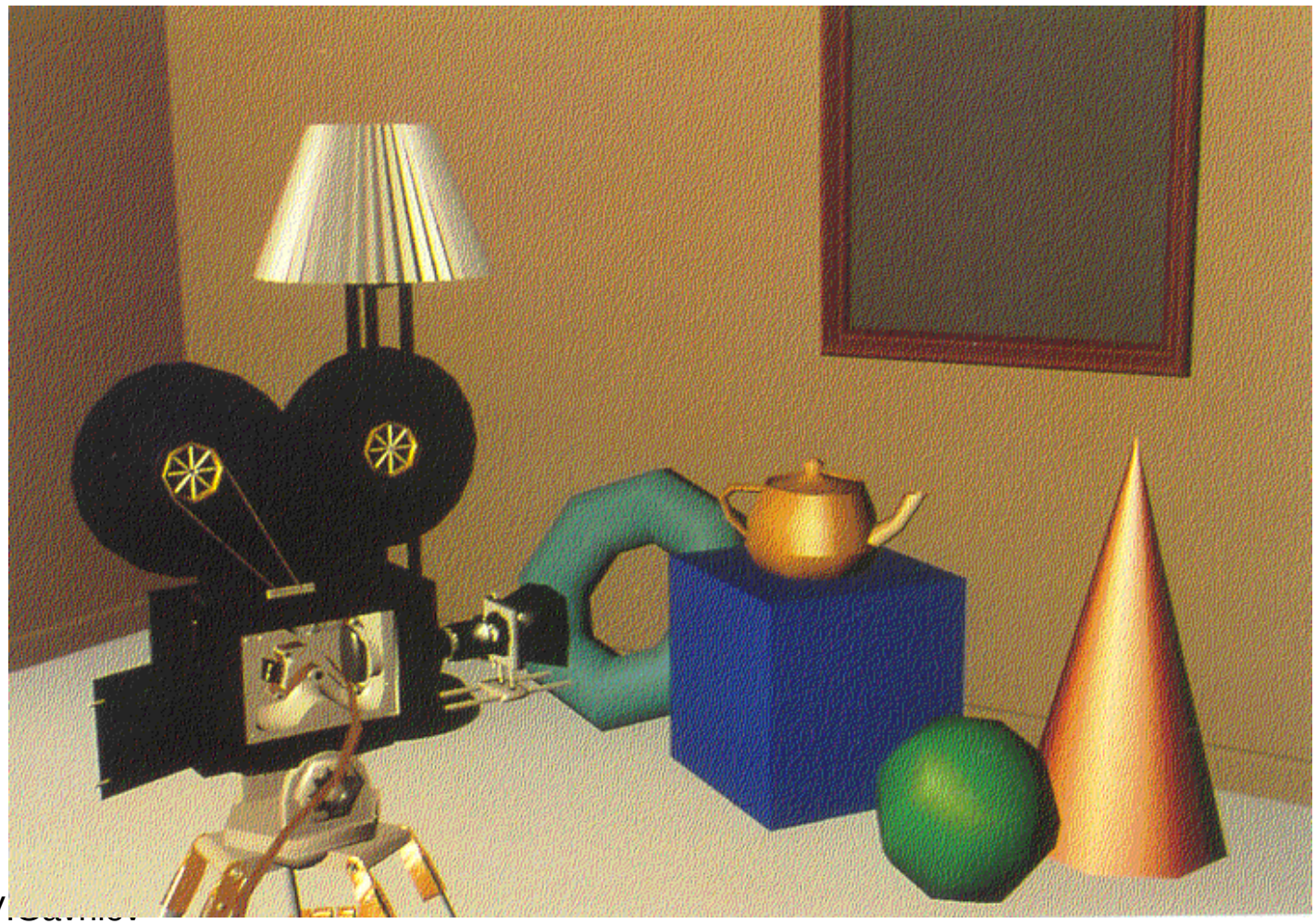
Flat filling



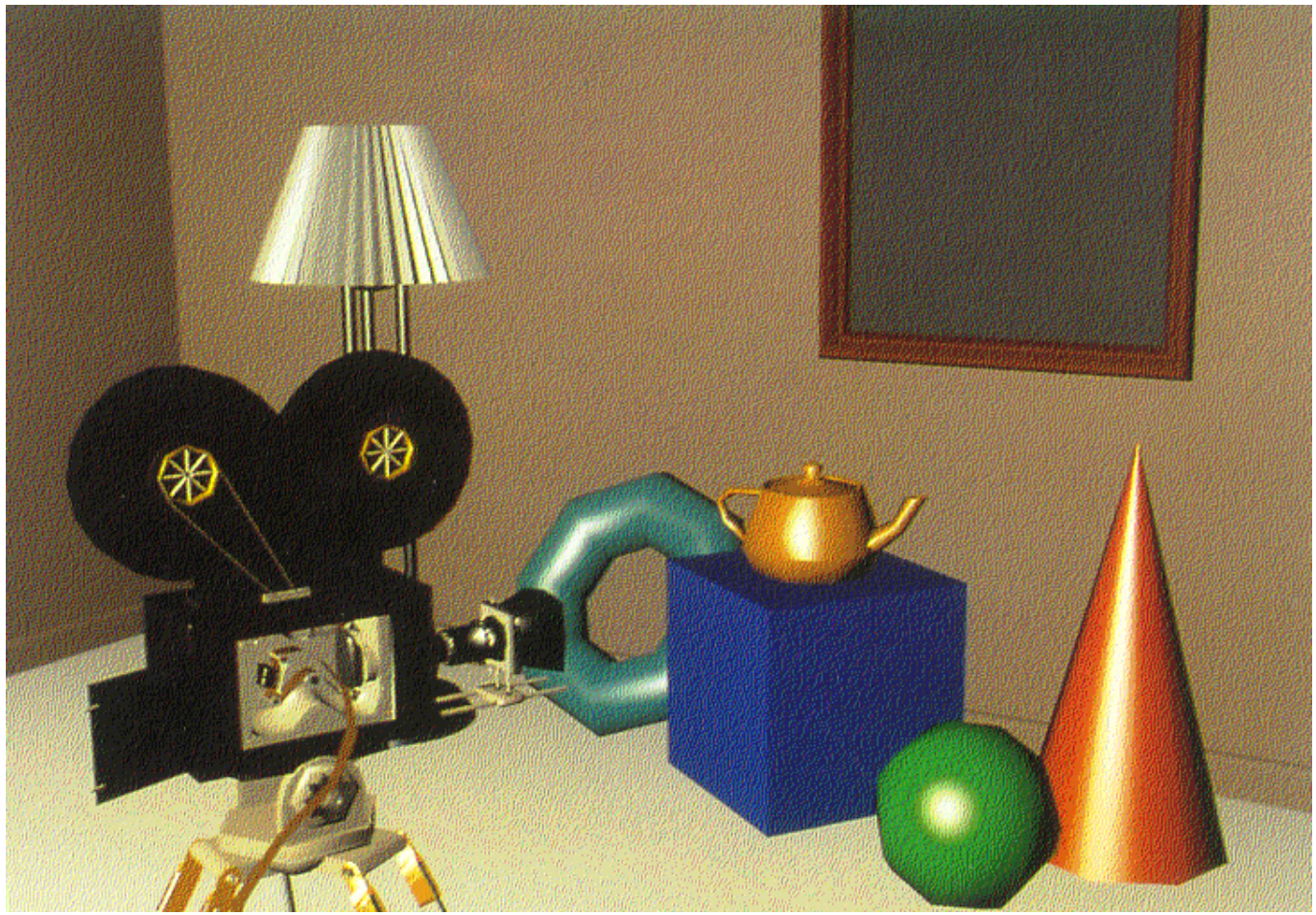
Gouraud filling (radiosity)



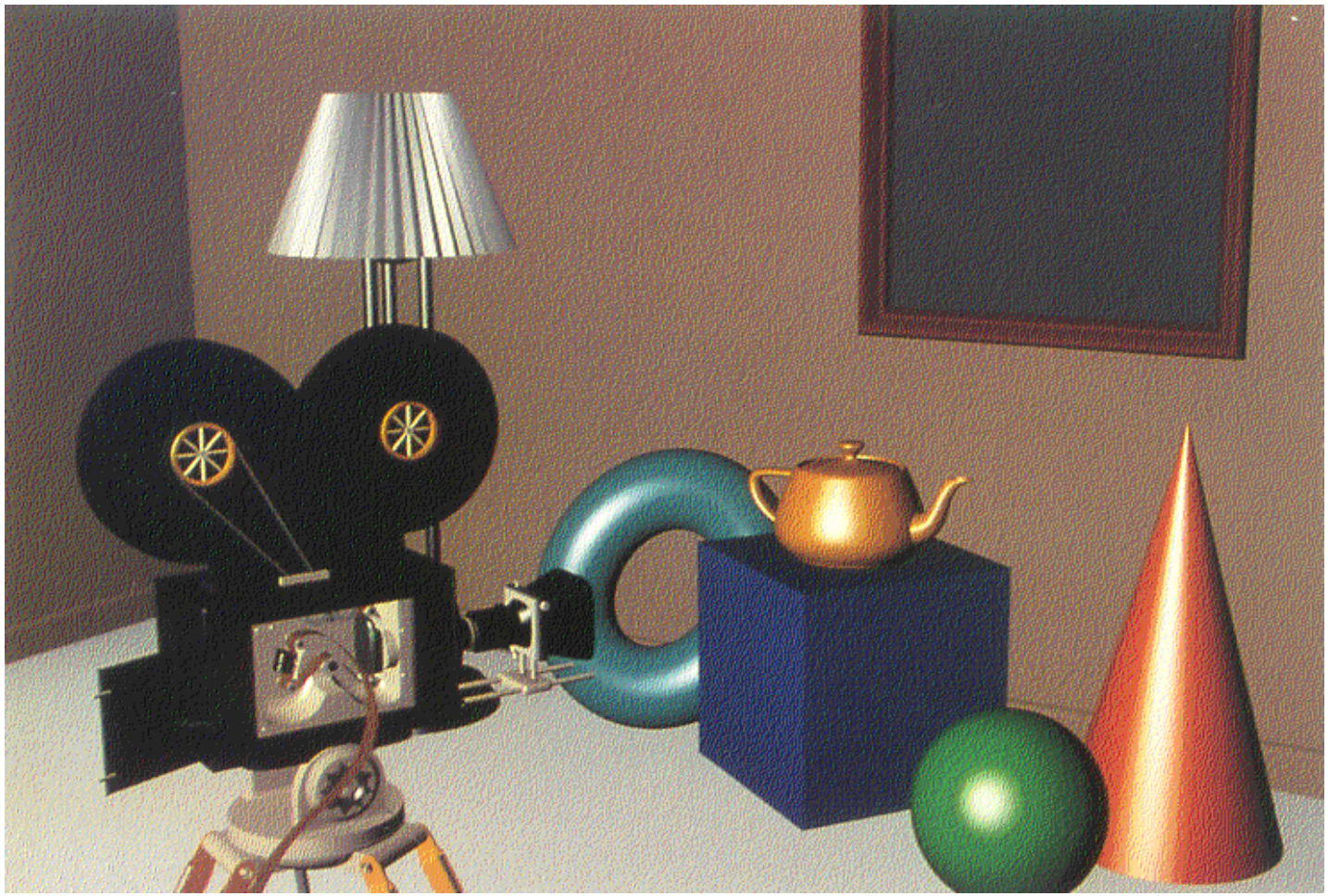
Gouraud filling (specular reflection)



Phong filling
(mirror reflection)



Curved surfaces (mirror reflection)



Textures and shadows (reflection in mirror)

Camera

- Camera is point in space where image is translating on screen
- This is really eye of player
- Screen is view plane
- Standard projection (standard camera in (0,0,0))

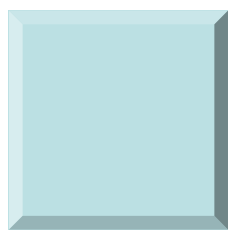
$$x` = x*FOV/z + xRes/2$$
$$y` = y*FOV/z + yRes/2$$

where x`, y` - coordinates on view plane;
x,y,z – coordinates of point in space;
xRes,yRes – resolution of screen;
FOV – angle of view of camera.

- To obtain projection for arbitrary camera it is needed to use matrix for transformation

Demo 1 (example 1 for chapter 5)

- This code example shows how to build the view matrix to follow a particular object.
- One thing to try: back up the object until the camera is looking right down on it. If you manage it, the screen will go to black. This happens because the cross product between the up vector and view vector goes to zero.
- Controls:
  - ESC: quit
  - I,J,K,L: translate object in the XY plane
  - SPACE: reset transforms



Routines

```
void
Game::UpdateObjects( float dt )
{
    // move view focus
    if (IvGame::mGame->mEventHandler-
        >IsKeyDown('k'))
    {
        mObjectPosition.x -= 3.0f*dt;
    }
    if (IvGame::mGame->mEventHandler-
        >IsKeyDown('i'))
    {
        mObjectPosition.x += 3.0f*dt;
    }
    if (IvGame::mGame->mEventHandler-
        >IsKeyDown('l'))
    {
        mObjectPosition.y -= 3.0f*dt;
    }
    if (IvGame::mGame->mEventHandler-
        >IsKeyDown('j'))
    {
        mObjectPosition.y += 3.0f*dt;
    }

    // reset to origin
    if (IvGame::mGame->mEventHandler-
        >IsKeyDown(' '))
    {
        mObjectPosition.Set( 0.0f, 0.0f, 1.0f );
    }
}
```

```
void
Game::LookAt( const IvVector3& eye, const IvVector3& lookAt,
const IvVector3& up )
{
    // compute view vectors
    IvVector3 view = lookAt - eye;
    view.Normalize();

    IvVector3 right =  view.Cross( up );
    right.Normalize();

    IvVector3 viewUp =  right.Cross( view );
    viewUp.Normalize();

    // now set up matrices
    // world->view rotation
    IvMatrix33 rotate;
    rotate.SetRows( right, viewUp, -view );

    // world->view translation
    IvVector3 xlate = -(rotate*eye);

    // build 4x4 matrix
    IvMatrix44 matrix(rotate);
    matrix(0,3) = xlate.x;
    matrix(1,3) = xlate.y;
    matrix(2,3) = xlate.z;

    ::IvSetViewMatrix( matrix );
}
```

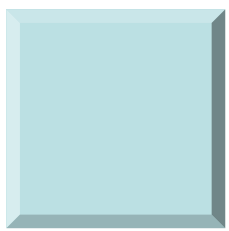
Rotines (2)

```
void
Game::Render()
{
    // set viewer
    LookAt( IvVector3(-10.0f, 0.0f, 10.0f), mObjectPosition, IvVector3::zAxis );

    // draw axes
    ::IvDrawAxes();

    // draw floor
    ::IvDrawFloor();

    // draw the object
    IvMatrix44 xform;
    xform.Translation( mObjectPosition );
    ::IvSetWorldMatrix( xform );

    ::IvSetColor( 1.0f, 0.0f, 1.0f );
    ::IvDrawTeapot();

}
```

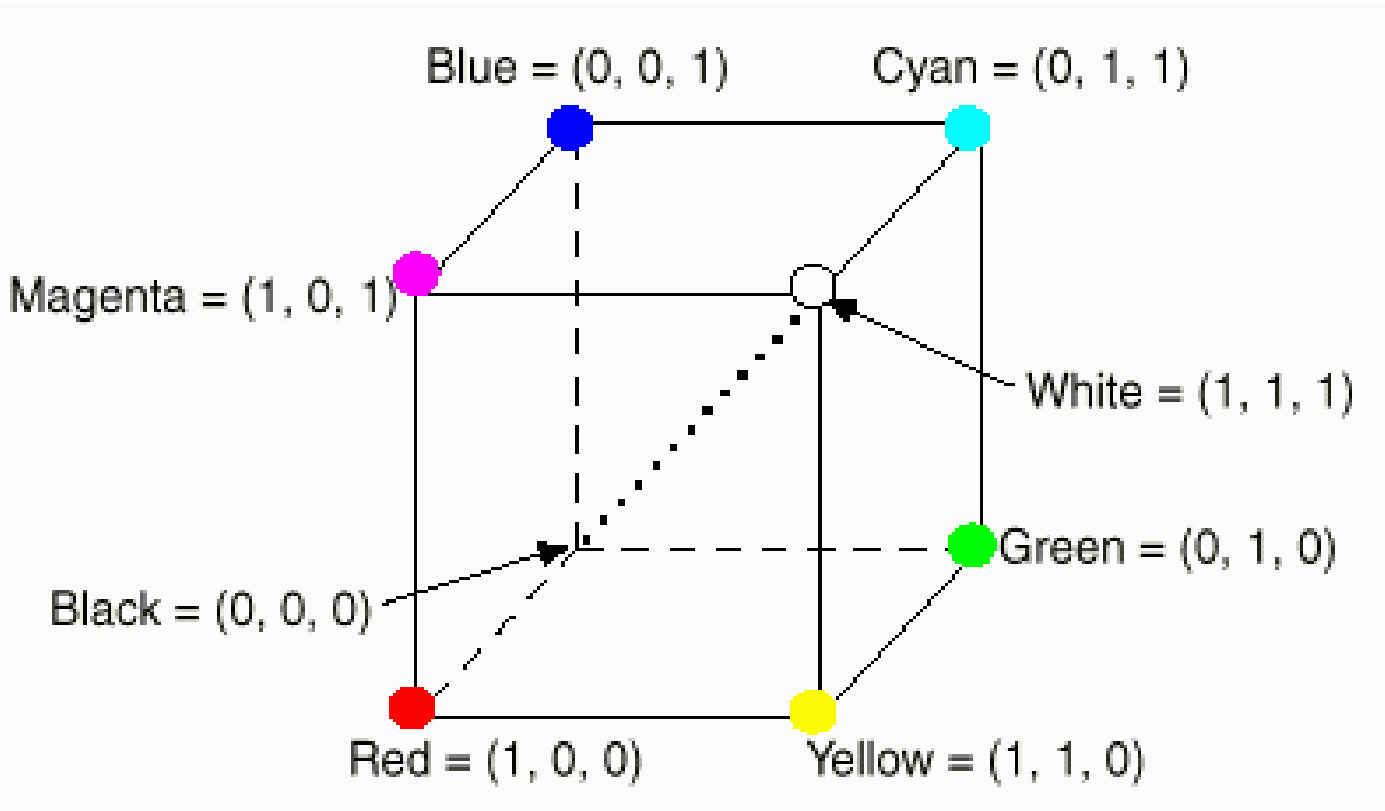
Demo 2 (example 2 for chapter 5)

- This code example shows how to build the view matrix based on a given rotation matrix and eye position.

- Controls
  - ESC: quit
  - i, k - rotate camera up and down
  - j, l - rotate camera left and right
  - space - reset to start



Color. RGB


Blue = (0, 0, 1)
Cyan = (0, 1, 1)
Magenta = (1, 0, 1)
White = (1, 1, 1)
Green = (0, 1, 0)
Black = (0, 0, 0)
Red = (1, 0, 0)
Yellow = (1, 1, 0)