From a developer's standpoint, it would be ideal to program the VR system by providing high-level descriptions and having the software determine automatically all of the low-level details. Which components will become more like part of a VR "operating system" and which will become higher level "engine" components? Given the current situation, developers will likely be implementing much of the functionality of their VR systems from scratch. This may involve utilizing a software development kit (SDK) for particular headsets that handles the lowest level operations, such as device drivers, head tracking, and display output. Alternatively, they might find themselves using a game en- gine that has been recently adapted for VR, even though it was fundamentally designed for video games on a screen. This can avoid substantial effort at first, but then may be cumbersome when someone wants to implement ideas that are not part of standard video games. of th Virt comp along force imag the c capt and the c then map What software components are needed to produce a VR experience? Figure 2.13 presents a high-level view that highlights the central role of the Virtual World Generator (VWG). The VWG receives inputs from low-level systems that indicate what the user is doing in the real world. A head tracker provides timely estimates of the user's head position and orientation. Keyboard, mouse, and game controller events arrive in a queue that are ready to be processed. The key role of the VWG is to maintain enough of an internal "reality" so that renderers can extract the information they need to calculate outputs for their displays. For example, if you drop an object, then it should accelerate to the ground due to gravitational force acting on it. One important component is a collision detection algorithm, which determines whether two or more bodies are intersecting in the virtual world. Developer choices for VWGs To summarize, a developer could start with a basic Software Development Kit (SDK) from a VR headset vendor and then build her own VWG from scratch. In this case, the developer must build the physics of the virtual world from scratch, handling problems such as avatar movement, collision detection, lighting models, and audio. Physics The VWG handles the geometric aspects of motion by applying the appropriate mathematical transformations. If the developer follows patterns that many before her have implemented already, then many complicated details can be avoided by simply calling functions from a well-designed software library. In addition, the VWG usually imple ments some physics so that as time progresses, the virtual world behaves like the real world. In addition to handling the motions of moving objects, the physics must also take into account how potential stimuli for the displays are created and propagate through the virtual world. Within the virtual world, user interactions, includ ing collisions, must be managed by the VWG. As applications of VR broaden, specialized VR engines are also likely to emerge. For example, one might be targeted for immersive cinematog- raphy while another is geared toward engineering design. User Locomotion In many VR experiences, users want to move well outside of the matched zone. These correspond to rendering problems, which are covered in Chapters 7 and 11 for visual and audio cases, respectively. Networked experiences In the case of a networked VR experience, a shared virtual world is maintained by a server. If multiple users are interacting in a social setting, then the burdens of matched motions may increase. The SDK should provide the basic drivers and an interface to access tracking data and make calls to the graphical rendering libraries. In a perfect world, there would be a VR engine, which serves a purpost similar to the game engines available today for creating video games. Simulated physics can become quite challenging. C C C Figu coule.