2
Trabalhos Relacionados

Neste capítulo são discutidos alguns trabalhos relacionados ao tema dessa pesquisa e que serviram como base para as soluções propostas neste trabalho.

2.1
Navegação em Ambientes Virtuais

A navegação em ambientes virtuais tem sido foco de estudo de vários pesquisadores. Boa parte desses trabalhos tem como objetivo principal propor e analisar técnicas que facilitem a tarefa de explorar esses ambientes.

Por exemplo, [32] analisam três diferentes ferramentas de navegação: *eyeball in hand*, *scene in hand* e *flying vehicle control*. Essas técnicas, apesar de inicialmente terem sido implementadas fazendo uso de dispositivos de controle para ambientes imersivos, podem ser alteradas para usar dispositivos mais comuns como mouse e teclado.

A primeira ferramenta, *eyeball in hand*, consiste em mapear os movimentos do dispositivo de entrada de forma a alterar a orientação da câmera virtual. A técnica *scene in hand* faz o oposto: movimentos realizados com o controle tem o efeito de alterar a orientação da cena, não da câmera. Esse comportamento é similar ao da ferramenta *examinar*, descrita na Seção 1.1. A última, *flying vehicle control*, corresponde à ferramenta *voar*, também descrita na Seção 1.1.

[32] concluem que nenhuma dessas três técnicas eram capazes de suprir, sozinhas, todas as necessidades dos usuários. Por exemplo, eles observaram que ao explorar ambientes internos, a ferramenta *flying vehicle control* é mais adequada, enquanto que *scene in hand* é mais indicada para ambientes externos.

[9] chegam à mesma conclusão ao analisar o comportamento de vários usuários. Ainda, observam também que boa parte das pessoas não têm compreensão de que geralmente são necessárias mais de um tipo de ferramenta para garantir uma navegação eficiente. Elas acabam sempre tentando usar uma mesma ferramenta para todos os fins. Como solução, os autores propõem

que os menus de seleção de ferramentas sejam organizados de forma a dar uma ênfase maior para aquelas que julgam ser mais importantes (Figura 2.1).

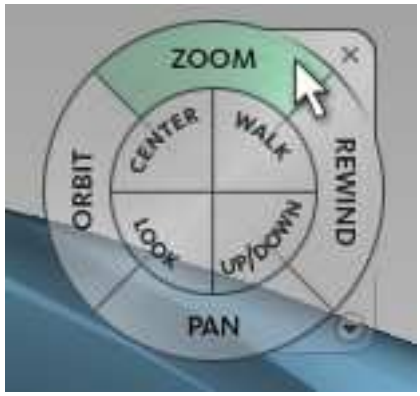


Figura 2.1: Organização do menu de ferramentas de interação [9].

[18] introduzem a técnica de *ponto de interesse (POI)*. Essa consiste em escolher um ponto da cena, onde o usuário deseja chegar. A partir daí a câmera realiza uma animação, seguindo de sua posição atual até o *POI*. A cada quadro, a posição da câmera é atualizada de acordo com a seguinte equação:

$$camPos = camPos - k(camPos - poi) \tag{2-1}$$

Na Equação 2-1 $camPos$ é a posição atual da câmera, poi é o ponto de interesse e k é uma constante que influencia na velocidade de aproximação. O efeito dessa equação é que a câmera parece se aproximar do poi a uma taxa proporcional à constante k. Segundo o autor, análises informais revelaram que esse tipo de movimento é mais natural ao usuário.

Em [27] é proposta uma solução híbrida, em que as ferramentas *voar* e *examinar* são compostas em uma única técnica de interação. O controle é feito com base no movimento de *dragging* do mouse: se o usuário inicia um *dragging* clicando em um ponto em que não há objetos, o comportamento resultante é o de *voar*. Se, ao contrário, o usuário clica em algum objeto, esse é posto no centro de tela e, a partir desse ponto, é possível examiná-lo.

[8] desenvolveram um sistema de controle automático de câmera baseado em restrições. Nele, a câmera sofre influência de várias restrições, definidas previamente com base no tipo de tarefa que se deseja realizar. Por exemplo, pode-se especificar que o vetor *up* da câmera sempre fique alinhado com o vetor *up* do mundo, ou que a câmera sempre aponte para um ponto específico do mundo. Um dos módulos desenvolvidos pelos autores permite determinar o caminho que a câmera deve seguir para ir de um ponto a outro. Para isso, um grafo de conectividade da cena é construido em tempo de pré-processamento e o caminho entre os dois pontos é calculado usando o algoritmo A*.

O sistema proposto por [8], entretanto, apresenta instabilidade no posicionamento de câmera. Como solução, [12] propõem que a câmera seja modelada por um sistema físico de partículas sujeitas a restrições de barras rigidas [14]. Isso permite movimentos mais suaves.

2.2 Ajuste dos Parâmetros de Navegação em Ambientes Multiescala

Ambientes multiescala foram inicialmente introduzidos pelos trabalhos de [23] e [4; 3], com a criação das interfaces *Pad* e *Pad++*. Esses sistemas permitiam visualizar informações em um plano de projeção 2D usando uma *lupa virtual*: o local da tela que é apontado por essa lupa é mostrado com mais detalhes. É possível também realizar operações de *zoom* nesses sistemas. Quando isso acontece, novas informações são exibidas, baseadas no contexto dos objetos de interesse. Por exemplo, no caso de um sistema de arquivos, inicialmente seriam vistas várias pastas. Ao dar *zoom* em uma dessas, seria possível ver os arquivos que estão dento dela e, no fim, o conteúdo de um determinado arquivo. Cada um desses níveis pode ser considerado uma escala diferente de visualização.

[10] desenvolveram o conceito de *diagramas de espaço-escala*, com o objetivo de facilitar a criação e análise de interfaces 2D multiescalas. Esses diagramas têm o formato de uma pirâmide de vários níveis, onde cada um desses representa uma escala referente a um ponto de vista do ambiente.

Apesar de serem basicamente interfaces 2D, os conceitos presentes nesses trabalhos podem ser aplicados também para a criação de ferramentas para navegação em ambientes virtuais 3D. [16], por exemplo, utilizam a mesma ideia de *lupa virtual* de [23].

O objetivo de [16] é permitir a exploração das diferentes escalas de um ambiente virtual que representa o corpo humano. Assim, o usuário deve ser capaz de visualizar o corpo, membros, orgãos, células e até o DNA. Para isso, os autores desenvolveram o conceito de *nível de escala* (*LoS*, do inglês *level of scale*).

Um *nível de escala* pode ser comparado com a ideia de *nível de detalhe* (*LoD - level of detail*). Em algoritmos de *LoD*, objetos próximos à câmera são mostrados com maior nível de detalhe, enquanto objetos distantes têm sua geometria simplificada, com o objetivo de melhorar a performance. Da mesma forma, um determinado *LoS* também mostra mais informações quando próximo da câmera. Entretanto, em um *LoS* não é somente o visual dos objetos que sofre alteração, mas também a forma como o usuário deve interagir e navegar nessa parte do ambiente.

[16] definem uma hierarquia de *LoSs*. Por exemplo, o corpo humano inteiro é a raiz da hierarquia e configura o primeiro *LoS*. Os membros do corpo vêm a seguir, cada um correspondendo a um novo *LoS*. Esse processo segue até o menor nível de escala possível, o DNA das células.

Para navegar entre esses diferentes níveis de escala, os autores usam uma *lupa virtual* para enxergar a existência de *LoSs* que estejam em hierarquias mais baixas (Figura 2.2). A troca de hierarquia é feita quando o usuário executa um comando e uma animação é então criada para indicar essa mudança. Uma vez que a câmera está no *LoS* de interesse, é possível usar a ferramenta *voar* para explorar essa parte do ambiente. A velocidade de navegação e os planos de corte são ajustados com a aplicação de um fator de escala dado pela seguinte equação:

$$fatorEscala = \sqrt[3]{\frac{Volume_{novoLoS}}{Volume_{antigoLoS}}} \tag{2-2}$$

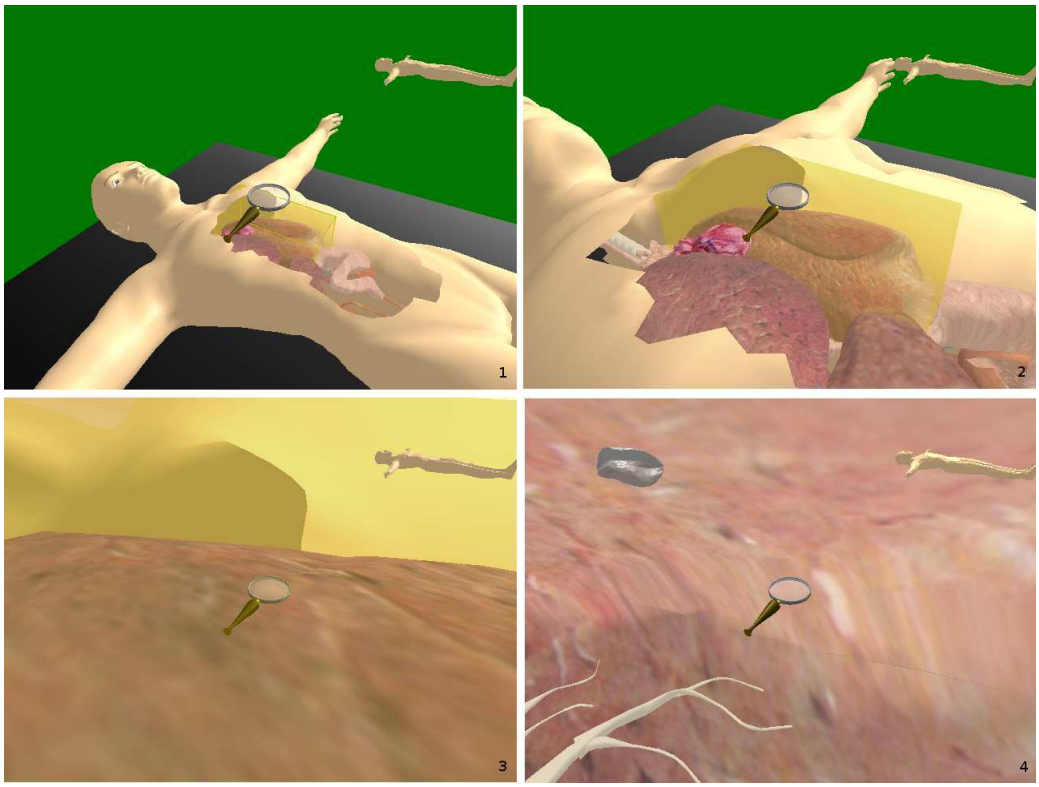


Figura 2.2: Visualização dos diferentes *níves de escala* do corpo humano com o auxílio da *lupa virtual* [16].

Na equação 2-2, $Volume_{novoLoS}$ e $Volume_{antigoLoS}$ são, respectivamente, os volumes das caixas envolventes do *LoS* que se deseja ir e do *LoS* anterior.

A técnica de [16] fornece uma maneira simples e funcional de exploração de diferentes níveis de escala. Entretanto, ela não permite a navegação livre entre dois diferentes *LoSs*. O usuário deve obrigatoriamente executar um

comando para realizar essa mudança. Por último, o ajuste dos parâmetros de navegação é feito somente no momento da mudança de escala. Dessa forma, pode-se dizer que o método de [16] é discreto no sentido de que os diferentes níveis de escala são bem definidos, tanto na sua forma quanto na sua localização na hierarquia de *LoSs*.

Os trabalhos de [31] e [19], ao contrário, realizam ajustes contínuos nos parâmetros de navegação e, dessa forma, não necessitam que o ambientes virtuais tenham uma hierarquia de níveis de escala bem definida.

[31] propõem ajustar a velocidade de navegação da ferramenta *voar* usando a informação de profundidade presente no *Z buffer*. Este buffer possui a mesma resolução da tela e guarda as distâncias de cada ponto dessa até a câmera. Por motivos de desempenho, o autor verifica apenas 15 linhas horizontais e igualmente espaçadas. Foram testados quatro diferentes formas de se ajustar a velocidade de navegação: usando o ponto mais próximo (menor valor de profundidade encontrado), o ponto mais distante (maior valor de profundidade), a média dos valores amostrados e por último, a média de valores amostrados considerando uma região de interesse. A conclusão a que os autores chegaram é que o ponto mais próximo é o mais adequado para realizar esse tipo de ajuste.

Essa também foi a conclusão de [19], que propõem uma ferramenta de navegação baseada em *POI* [18]. Nesse trabalho, a distância da câmera ao ponto mais próximo da cena é usada para controlar a velocidade de navegação de acordo com a seguinte equação:

$$camPos = camPos + (poi - camPos)t_{\Delta}\frac{minDist}{2} \qquad (2\text{-}3)$$

Na equação 2-3, $camPos$ é a posição da câmera, poi é o ponto de destino, t_{Δ} é o intervalo de tempo entre dois quadros de animação e $minDist$ é a distância da câmera ao ponto mais próximo da cena.

Para determinar $minDist$ é construído um *cubo de distâncias*, cujas faces correspondem aos 6 planos de projeção resultantes da orientação da câmera nas 6 direções canônicas. Cada face guarda informação de profundidade, similar ao *Z buffer*. O cálculo dessa estrutura é feito da seguinte forma: toda vez que a câmera se move, as faces do cubo são renderizadas em 6 imagens diferentes, usando a nova posição da câmera. O ângulo de abertura da câmera é de 90^0, de forma a cobrir todo o espaço do ambiente virtual, sendo limitado apenas pelos planos de corte. Através de um programa que roda na placa gráfica, a distância de um ponto visível até câmera é escrito no canal alpha do pixel referente ao fragmento gerado pelo ponto. Ao final, $minDist$ é determinado fazendo uma varredura pelo menor valor presente nas imagens. Por uma questão de

desempenho, a renderização do cubo de distâncias é feita usando uma resolução menor do que a da aplicação.

Além de ajustarem a velocidade de navegação, [19] também utilizam *minDist* para determinar os planos de corte. No Capítulo 3, o modo como isso é feito é explicado, assim como a construção do cubo de distâncias é descrito com mais detalhes.

2.3 Detecção e Tratamento de Colisão

[2] classificam as técnicas de detecção de colisão entre objetos em ambientes virtuais em duas linhas principais de pesquisa, com base no tipo de teste de interseção que é feito: *testes de interseção feitos no espaço do objeto* e *testes de interseção feitos no espaço da imagem*.

As técnicas baseadas em testes feitos no espaço do objeto usam diretamente a informação de geometria dos modelos para verificar a existência de colisão entre esses. Geralmente é necessário pré-computar estruturas de dados especiais, como árvores BSP [28], que permitam detectar regiões de possível colisão. O desempenho desses métodos é relacionado com a quantidade de geometria existente: quanto maior o número de modelos, maior será o custo envolvido no cálculo de colisão.

As técnicas baseadas no espaço da imagem consistem em projetar a geometria no plano de imagem usando o hardware gráfico e, então, são realizados testes de interferência geralmente através da análise do *Z buffer* [1; 2]. Uma vez que os testes são feitos no espaço da imagem, a quantidade de geometria tem uma influência bem menor sobre o custo para computar a colisão. Outro ponto positivo é que esses métodos fazem uso maior do processamento das placas gráficas, reduzindo o número de cálculos realizados na CPU.

Os trabalhos mencionados anteriormente se preocupam com a detecção de colisão entre todos os objetos da cena. Entretanto, o problema que se quer tratar aqui é um pouco mais simples: deseja-se detectar se há colisão somente entre a câmera e os modelos da cena, com o objetivo de melhorar a experiência de navegação para o usuário.

Diversas técnicas foram propostas nesse sentido. Por exemplo, para permitir uma navegação eficiente em modelos de reservatórios de petróleo, [5] controem um grafo de conectividade em tempo de pré-processamento que os permite traçar um caminho livre de colisão para a câmera. Nesse grafo, os nós representam locais em que a câmera pode ir sem que haja colisão. As arestas indicam a existência de um caminho, também livre de colisão, entre os

nós conectados por essas.

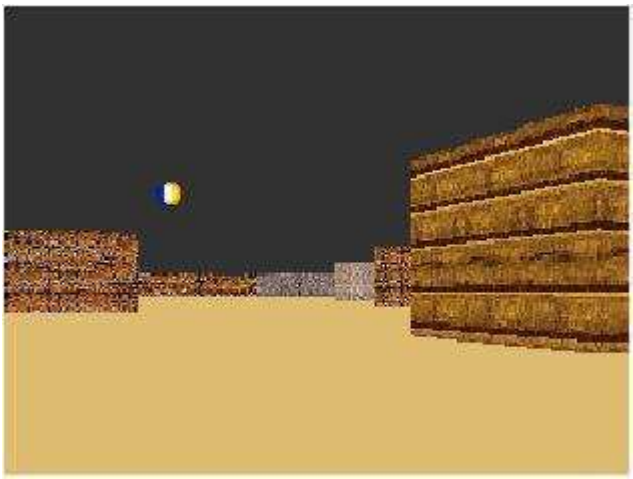
Com base nessa estrutura, [5] desenvolveram dois modos de navegação:

- *Modo de navegação automático*: nesse modo, o usuário seleciona um nó de destino e a câmera realiza uma animação livre de colisão entre o ponto atual e o de destino. A escolha do caminho é feita usando uma versão modificada do algoritmo A*.
- *Modo de navegação assistida*: nesse modo o usuário pode navegar livremente pelo ambiente usando a ferramenta *voar*. Entretanto, o grafo de conectividade é consultado diversas vezes durante a navegação para verificar a existência de colisão. Com base na direção de movimento da câmera, um nó de destino é inferido. O nó de origem é determinado a partir da posição atual da câmera. O algoritmo A* é então executado para achar um caminho entre esses dois nós, que por sua vez será usado para corrigir a trajetória da câmera de forma a evitar colisões com o ambiente.

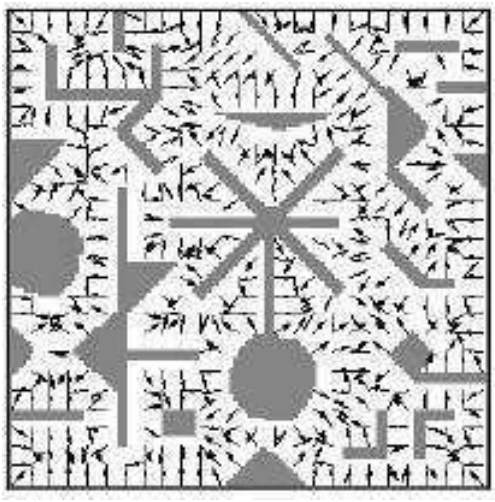
Esse tipo de navegação assistida também foi usada por [33]. Em sua pesquisa, ele introduz o uso de *campos de forças artificiais* no processo de navegação. Nessa abordagem, os objetos são envoltos por forças de repulsão. Essas são calculadas durante a navegação, através de raios lançados contra a geometria, a fim de encontrar os objetos mais próximos. Ao se aproximar de um modelo, a câmera é repelida por uma dessas forças. Quanto maior a proximidade da câmera, maior é a força de repulsão. Como efeito, a câmera sofre um desvio no sentido de ser jogada para fora da região de colisão.

[17] estendem a ideia de [33] e transferem a computação das forças para uma etapa de pré-processamento. Para isso, eles assumem que o ambiente virtual pode ser representado em um formato 2D (Figura 2.3). Como resultado, é possível obter um mapa de campo de forças com maior resolução, se comparado a técnica de [33]. Durante a navegação, a amplitude desses vetores de forças são alterados com base na posição da câmera, de forma o obter o mesmo efeito de [33].

Ainda na linha de pesquisa baseada em campos de forças artificial, [19] propõem o uso da placa gráfica para auxiliar na construção do mapa de forças. Para isso, ele usa os canais RGB do cubo de distâncias para guardar o vetor que aponta na direção da câmera, vindo do ponto referente ao fragmento gerado no processo de renderização. Essa abordagem permite obter uma mapa de forças com resolução maior, além de eliminar a etapa de pré-processamento proposta por [17]. A desvantagem é que a construção do cubo de distâncias aumenta



2.3(a):



2.3(b):

Figura 2.3: Mapa de campo de forças: em (a), o ambiente virtual de um labirinto. Em (b), a representação 2D desse labirinto, juntamente com os vetores do campo de força. Figuras retiradas de [17].

o processamento necessário durante a navegação, uma vez que são realizadas mais 6 passadas de renderização.

O mapa de forças presente no cubo de distâncias é utilizado para calcular uma força de repulsão resultante, que por sua vez é usada em conjunto com a ferramenta estilo *POI* descrita na Seção 2.2. Essa força faz com que a câmera seja repelida pelos objetos encontrados no meio do caminho.

2.4 Auxílio para Orientação e Localização

Além dos problemas discutidos anteriormente, outro motivo frequente de navegação ineficiente tem relação com a dificuldade que alguns usuários têm em se orientar e se localizar em ambientes virtuais.

Dentre as diversas causas, [20] cita a inexistência de informações suficientes e o projeto ruim das ferramentas de navegação. Essa última causa também é apontada por [9], que identifica a ocorrência do problema descrito na Seção 1.2.4, relativo ao posicionamento do centro de rotação da ferramenta *examinar*.

Os autores observaram que os usuários tinham dificuldade em lidar com o centro de rotação pois muitos ignoravam, ou mesmo não percebiam, sua existência. Como solução, eles decidiram mostrar uma pequena esfera para indicar a localização desse. Além disso, eles estabeleceram que o centro de rotação sempre deve estar visível na tela.

A técnica anterior ilustra um tipo de solução muito comum em ambientes virtuais: se naturalmente não há informação visual sufuciente, então ela deve ser criada no sentido de guiar o usuário. Por exemplo, [6] permitem que os usuários deixem marcas pelo caminho enquanto estão navegando, a fim de que possam refazer o caminho de volta e não fiquem perdidos.

[15] introduzem o conceito de *resíduo*, que possibilita que o usuário perceba a existência de um objeto, mesmo que este não esteja visível por estar muito pequeno. A ideia é colocar marcas onde os objetos deveriam estar aparecendo. Essa também foi a abordagem aplicada por [24] em seu trabalho, no qual é descrito um sistema de navegação baseado em marcas com formato dependentes dos objetos que elas representam (Figura 2.4). Por exemplo, um farol é representado por um ícone na forma de um farol em miniatura. Esses ícones só se tornam visíveis ao usuário quando este está afastado a uma certa distância. Para evitar que essas marcas se sobreponham, criando assim um emaranhado confuso, os autores também propõem técnicas para decidir quando e quais ícones devem ser visíveis em um dado instante.

[26] criaram uma versão em miniatura do mundo virtual (Figura 2.5). Através dela é possível ao usuário saber sua localização e orientação, e este pode planejar melhor qual caminho seguir, uma vez que é possível visualizar todo o ambiente através da miniatura. O modelo também pode ser rotacionado, a fim de se conseguir ter vários pontos de vista do ambiente. As ações realizadas no modelo em miniatura não afetam o de escala real. Dessa forma, ações incorretas executadas pelo usuário podem ser facilmente revertidas.

Por último, a indústria de jogos eletrônicos também fornece vários exemplos de técnicas que visam facilitar a orientação e localização em mundos virtuais. Muitos jogos de estratégia usam mapas 2D e marcam neste a posição do jogador. Jogos de corrida também fazem uso dessa técnica e, além da posição do carro, indicam também a direção que deve ser seguida através de setas. Essas também são usadas para indicar a posição de um determinado objeto da cena, ou mesmo outro jogador no caso de jogos *multiplayers*. Essa ideia

2.4(a):

2.4(b):

Figura 2.4: Visualização de uma fazenda antes (a) e depois (b) da adição de marcas para os objetos da cena [24].

foi incorporada ao SiVIEP com a criação de uma *seta indicadora*, a qual é explicada com mais detalhes no próximo capítulo.

2.5 Análise Final

Analisando os diferentes trabalhos mencionados anteriormente é possível estabelecer um conjunto de técnicas que melhor se adequam aos requisitos do SiVIEP.

A solução proposta por [19] é interessante pois trata de três dos problemas apresentados aqui: velocidade de navegação, ajuste dos planos de corte e detecção e tratamento de colisão. As técnicas são baseadas em uma estrutura chamada de *cubo de distâncias*, cuja construção independe do formato ou das características dos objetos da cena. Também não é necessário que nenhuma etapa de pré-processamento seja realizada.

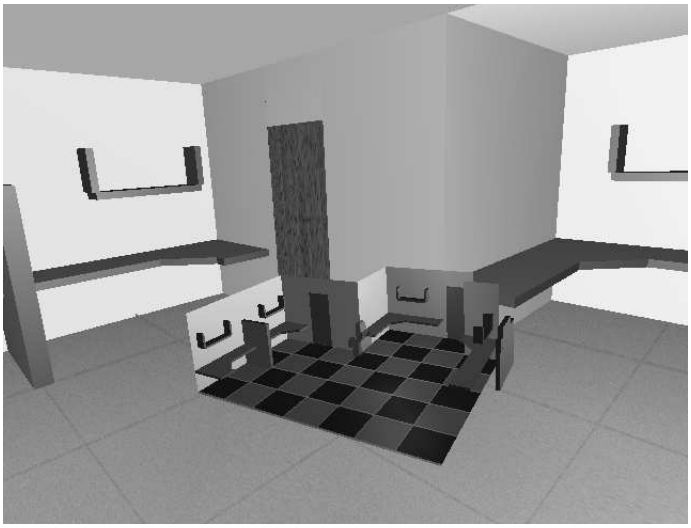


Figura 2.5: Visualização da versão em miniatura (centro da figura) de um cenário virtual [26].

As técnicas discutidas em [16] também não requrem pré-processamento, mas necessitam que uma hierarquia de níveis de escala possa ser bem definada. Isso gera um novo problema, no qual a questão é como a hierarquia deve ser estabelecida. Ainda, a navegação exige que o usuário conheça e compreenda as relações existentes entre os diferentes níveis de escala.

O modelo de navegação assistida de [33], [17] e [5] pode ser implementado no SiVIEP em conjunto com a ferramenta *voar* já existente e com a detecção de colisão. O grafo de conectividade descrito em [5] permite também determinar a trajetória entre dois pontos quaisquer do ambiente. Entretanto, a construção desse grafo exige pré-processamento e é dependente da forma da geometria que os autores se dispõem a visualizar.

Como conclusão observou-se que, dentre as técnicas estudadas, aquelas propostas por [19] são as que melhor atendem ao SiVIEP. Em especial, o *cubo de distâncias* se revelou útil também na criação de uma solução automática para o problema do centro de rotação, e de uma técnica cujo objetivo é auxiliar o usuário a se localizar e orientar no ambiente virtual. Os detalhes do funcionamento e de como essas técnicas foram implementadas no SiVIEP são tema do próximo capítulo.