UNIVERSIDADE FEDERAL DO PAMPA

Leonardo Brondani Gonçalves

Geração de modelos 3D a partir de esboços de plantas baixas

Alegrete
2019

Leonardo Brondani Gonçalves

Geração de modelos 3D a partir de esboços de plantas baixas

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 24 de junho de 2019
Banca examinadora:

Prof. Dr. Marcelo Resende Thielo
Orientador
UNIPAMPA

Prof. Me. Aldo Leonel Temp
UNIPAMPA

Prof. Dr. Claudio Schepke
UNIPAMPA

Prof. Me. Eliezer Soares Flores
UNIPAMPA

RESUMO

Construção civil é o ramo que trabalha com confecção de obras, levando em consideração os custos e buscando maior eficiência. Um dos documentos criados no início de um projeto de construção civil é a planta baixa, que é uma representação do que será construído. Este trabalho apresenta o desenvolvimento de uma ferramenta capaz de criar, a partir do esboço de uma planta baixa, um modelo 3D do que será construído, incluindo o mobiliário de forma automática, com a finalidade de visualização. A ferramenta proposta utiliza recursos de visão computacional para detectar objetos em uma imagem. A entrada do programa consiste em uma imagem de uma planta baixa feita à mão, a partir da qual será realizada uma detecção inicial de retas. As retas representam as paredes e aberturas e, a partir dos vértices dessas retas, serão construídas tridimensionalmente as paredes da construção. Após criar as paredes do modelo, a ferramenta irá posicionar móveis estrategicamente em seus devidos cômodos com base no tamanho da área de cada cômodo. Esperamos que a ferramenta possa ajudar o profissional civil a entender melhor seu projeto, e também a mostrar de forma mais rápida o resultado final aproximado aos seus clientes.

Palavras-chave: Planta baixa. Modelo 3D. Construção civil. Visão computacional. Computação gráfica.

ABSTRACT

Civil construction is the branch that works with the construction of works, taking into account the costs and seeking greater efficiency. One of the documents created at the beginning of a civil building design is the ground, which is a representation of what will be built. This work presents a tool capable of creating, from the sketch of a low plan, a 3D model of what will be built. The proposed tool uses computer vision techniques to detect objects in an image. The input of the program is a picture of a low floor plan drawn by hand, and from it will be performed a sensing of the ground. The straight lines represent the walls and openings and, from the vertices of these, the walls of the building will be constructed three-dimensionally. After creating the walls of the model, the tool will position furniture strategically in its proper rooms. We expect that the tool will help the construction professionals to better understand his project, and also to show the final result more clearly to his clients.

Key-words: Floor plan. 3D model. Construction. Computer vision. Computer graphics.

SUMÁRIO

1 INTRODUÇÃO

Construção civil é um termo que engloba a confecção de obras como edifícios, casas, pontes, etc. O processo de planejamento e construção de obras civis tem tido um grande impacto tecnológico de forma consistente e revolucionária, sempre buscando obter mais agilidade no processo de construção e redução de custos. Desde o início da história o homem já construía seu próprio abrigo com os materiais que tinha à sua volta e, com o passar dos anos, essas estruturas foram se desenvolvendo e ficando mais complexas. Hoje em dia um projeto de construção civil já é algo bem tecnológico, e envolve inúmeros elementos como, por exemplo, planta baixa, planta de locação, planta de cobertura, quadro de esquadrias, projeto estrutural, projeto elétrico, projeto hidráulico, memorial descritivo dos materiais, entre outros. Para isso são necessários vários profissionais para criar os elementos já mencionados e muitos outros.

Neste trabalho foi desenvolvido uma série de métodos e algoritmos para que, utilizando apenas o esboço de uma planta baixa desenhada à mão, a qual é uma visualização de cima do que será construído, obter suas delimitações, estimando a localização de cada cômodo. A planta baixa constitui a primeira forma do projeto e, a partir da análise deste documento, serão construídas virtualmente as paredes e as divisórias da construção, a fim de visualizar o projeto em uma perspectiva tridimensional. Além disso, a ferramenta proposta neste trabalho também irá posicionar, com base na estimativa dos cômodos, o mobiliário correspondente de forma adequada. A estimativa gerada não garante sem chance de erro a descoberta dos cômodos, mas ajuda o engenheiro ou arquiteto a construir uma primeira visualização, facilitando o seu trabalho e acelerando uma série de aspectos do projeto de um imóvel. Com esse tipo de visualização, espera-se que o profissional de engenharia ou arquitetura tenha mais facilidade para entender seu projeto, e também possa mostrar de uma forma mais rápida e visual o resultado final aos seus clientes.

Uma série de trabalhos já foram publicados realizando a geração de plantas digitais automaticamente a partir de desenhos no papel, mas elas normalmente se baseiam em plantas criadas por profissionais ou então desenhadas com instrumentos como régua e esquadro, exigindo considerável rigor e precisão na planta utilizada como entrada dos algoritmos apresentados por esses autores.

1.1 Objetivo

O objetivo deste trabalho consiste na criação de uma ferramenta que auxilia na visualização de projetos civis de uma forma rápida e simples. A ferramenta irá permitir que o usuário crie um modelo tridimensional do seu projeto a partir de uma fotografia de uma planta baixa desenhada à mão livre, contendo apenas a representação das paredes da construção, sem a necessidade de grande precisão nos traços. Após a geração das paredes e aberturas da construção, a ferramenta também definirá uma organização de móveis de acordo com o tipo de cômodo (quartos: cama, roupeiro..., cozinha: geladeira,

fogão...), o qual será determinado com base no tamanho da área de cada cômodo. Um outro ponto forte da ferramenta será no sentido de facilitar a comunicação do profissional civil com seus clientes. Quando pessoas contratam um arquiteto ou engenheiro civil por exemplo, muitas vezes não têm uma visão clara do que procuram apenas olhando a planta baixa do projeto. Com o auxílio da ferramenta, será possível mostrar de uma melhor forma o edifício, possibilitando modificações no projeto e reduzindo a chance de possíveis frustrações.

1.2 Organização deste trabalho

Nesta seção é descrita a maneira como este documento foi organizado.

O Capítulo 2 mostra os fundamentos teóricos utilizados para o trabalho, que estão relacionados a construções civis, tecnologias 3D e tecnologias para reconhecimento de imagens.

O Capítulo 3 contém informações de como foi realizada a busca e a seleção de trabalhos relacionados. Além disso, neste capítulo, está contida a descrição dos trabalhos selecionados.

O Capítulo 4 descreve como será feito o trabalho em si, mostrando as etapas que serão realizadas ao longo do desenvolvimento.

O Capítulo 5 mostra os resultados obtidos de detecção e correção de retas, geração da paredes 3D da planta e posicionamento dos móveis nos cômodos.

O Capítulo 6 apresenta as conclusões do desenvolvimento do trabalho.

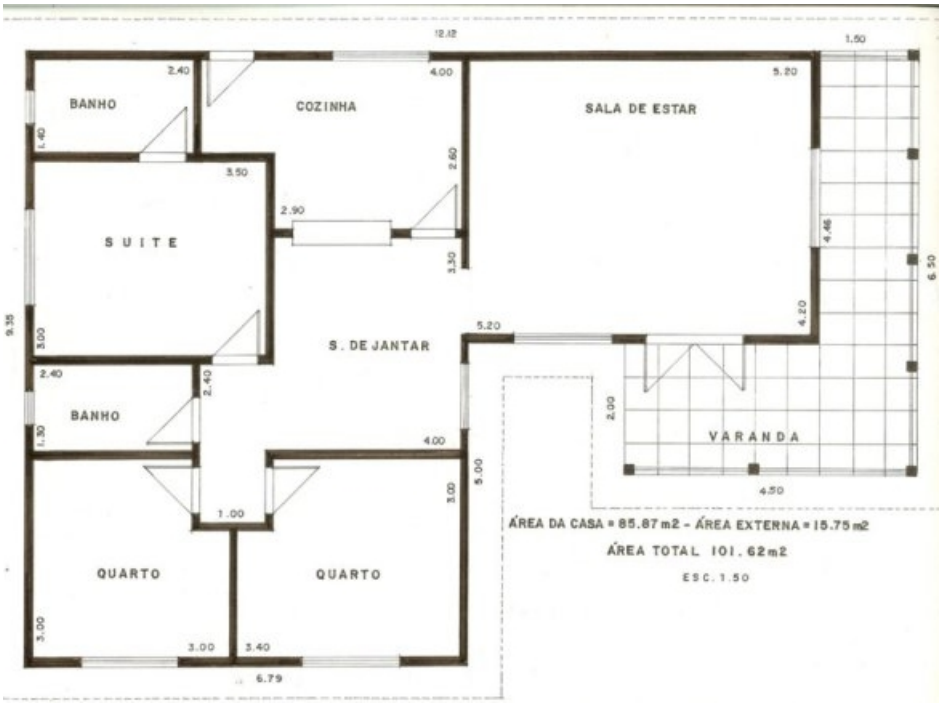
2 FUNDAMENTAÇÃO TEÓRICA

Serão apresentados nesta seção os fundamentos teóricos utilizados para realização deste trabalho: Visão Computacional, *OpenCV* e planta baixa.

2.1 Planta baixa

Projetos de construções civis são bem complexos e necessitam de diversos documentos e profissionais para criá-los. Um desses documentos é a planta baixa, que é uma representação do que será construído. A planta baixa é um desenho técnico feito a partir do corte horizontal da construção, à uma altura de 1,5m. É um diagrama que representa o relacionamento entre as salas, espaços e outros aspectos físicos. Neste documento também deve conter outras informações, como espessura e comprimento das paredes, aberturas (portas e janelas) e também uma identificação que irá definir o que será cada um dos cômodos da construção como podemos observar na Figura 1. Pode também haver uma representação de móveis na planta, para dar uma noção de tamanho dos cômodos.

Figura 1 – Planta baixa.



Fonte: (SANTOS, 2015)

2.2 Visão computacional

Segundo Learned-Miller (2011), "Visão computacional é a ciência de dotar computadores ou outras máquinas com visão ou a capacidade de ver". No entanto, neste

contexto, o conceito de visão não é simplesmente "ver". Visão é a percepção do mundo exterior, que pode levar à uma ação ou decisão. Um exemplo disso é um animal em busca de alimento, o qual utiliza sua visão para diferenciar o que é comestível e o que não é, ou então perceber a presença de predadores seja através de sinais (pegadas, evidências, etc) ou por contato visual direto.

Às vezes, a interpretação de uma imagem pode levar a ações imediatas como, por exemplo, se esquivar de algo. Porém, não podemos descartar o simples ato de desfrutar uma paisagem, pois os processos de reconhecimento, interpretação e aprendizado podem estar ocorrendo naquele momento, e no futuro uma decisão pode ser baseada nas memórias daquela visão. Muitos aspectos são importantes para extrair características de uma imagem. Para ter uma noção de profundidade, cada olho capta uma imagem em um ponto de vista ligeiramente diferente do outro, e com isso o cérebro combina essas imagens formando uma imagem tridimensional, dando uma percepção de profundidade. Há também questões de concluir se uma dada superfície é úmida, transparente ou reflexiva. Isso é possível a partir de pequenos realces que a visão consegue captar na imagem.

A visão computacional estuda como seria possível criar uma máquina que seja capaz de fazer inferências sobre o mundo externo a partir de imagens, tentando chegar o mais próximo possível das características de visão discutidas anteriormente. Dado um conjunto de imagens, quais delas possuem quadrados e quais possuem cubos? A máquina deveria levar em conta muitos aspectos, como perceber se o objeto possui profundidade (indicando um cubo) a partir de observações da diferença do brilho em relação a luz, observar também as sombras, entre outras características necessárias para fazer essa distinção. entretanto nunca poderemos ter certeza absoluta do resultado.

2.2.1 Tarefas típicas

Na visão computacional existem algumas tarefas típicas, que são realizadas no processo de reconhecimento de imagens de forma robusta e sem esforço humano.

- **Reconhecimento** - O reconhecimento de objetos, características ou um evento em imagens é o problema clássico da visão computacional e do processamento de imagens. Para nós, humanos, o problema é resolvido de forma robusta e sem esforço. Porém, para máquinas, ainda não existe solução satisfatória para o caso de reconhecimento de objetos em geral. Atualmente, os métodos conseguem apenas resolver problemas específicos como reconhecer formas geométricas, faces humanas, caracteres ou veículos, também com a necessidade de características específicas do ambiente bem definidas, como iluminação, fundo fixo, ângulo de visão dos objetos e qualidade em geral da imagem.

  - **Detecção** - Digitaliza a imagem para uma condição específica, para então fazer uma diferenciação de objetos.

  - **Reconhecimento** - Análise de uma ou mais classes pré-definidas (com um conjunto de posições em imagens 2D ou com sua pose em imagens 3D).
  - **Identificação** - Reconhecimento de uma instância individual de objeto, como por exemplo identificação de faces ou impressões digitais.

- **Restauração de imagens** - O objetivo desta tarefa é simplesmente reparar imagens a partir de remoção de ruídos.

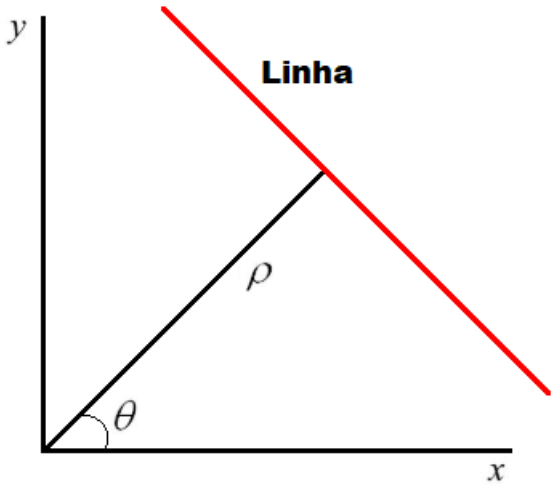
- **Movimento** - Esta tarefa consiste em analisar uma sequência de imagens a fim de produzir uma estimativa da velocidade em cada ponto.

- **Reconstrução de cena** - A reconstrução de cena é uma tarefa que visa reconstruir um modelo tridimensional da cena a partir de um ou mais imagens ou um vídeo. Nos casos mais simples consiste em reconstruir apenas a posição dos objetos na cena, e em casos mais sofisticados são também reconstruídas texturas e cores dos objetos e ambiente.

2.3 Transformada de Hough

Transformada de Hough é uma técnica matemática utilizada na análise de imagens, visão computacional e processamento digital de imagens. Foi criada por Hough (1962) e, em sua concepção original, a técnica se preocupava com a detecção de retas em uma imagem. A Transformada de Hough detecta grupos de *pixels* que pertencem a um reta, mesmo que a reta esteja quebrada ou com ruídos. Este método consegue fazer isso da seguinte maneira: uma linha geralmente é representada por: $y = mx + b$. Onde m é a inclinação da reta e b é a intersecção. Dessa forma, uma reta pode ser representada por um ponto (b, m). No entanto, os dois parâmetros são limitados, pois quando a reta se torna vertical, as magnitudes de m e b tendem ao infinito. Para resolver isso, é feita uma parametrização nas retas usando dois outros parâmetros (θ, ρ), onde: $\rho = x\cos(\theta) + y\sin(\theta)$. Dessa forma cada reta se associada a um único ponto (θ , ρ) no espaço de Hough. E dado um ponto (θ, ρ), o conjunto de todas a retas que cruzam esse ponto irão formar uma curva senoidal. Um conjunto de dois ou mais pontos de formam uma reta irá produzir senóides naquele ponto (θ, ρ) para aquela linha. Qualquer linha pode ser representada por (θ, ρ). Para reconhecer as linhas é criado então um contador definido inicialmente em 0. Depois, o programa irá pegar o primeiro ponto (x, y) da linha. Então, na equação da reta é utilizado o $\theta = 0, 1, 2, 3, \ldots\ n$ ($n = 180$ para precisão dos ângulos seja de grau 1) e verificar o ρ obtido. Para cada par (θ, ρ) é incrementado em 1 o contador em sua célula correspondente. No final, basta procurar o contador com valor máximo, que irá corresponder a um par (θ, ρ), que representa uma linha que fica a ρ de distância da origem e possui um ângulo de θ graus.

Figura 2 – Gráfico que representa uma linha (θ, ρ) .



Fonte: o autor

2.4 Computação gráfica

A computação gráfica é uma área da ciência da computação que se dedica ao estudo de técnicas capazes de manipular imagens através de uma máquina (MANSSOUR; COHEN, 2006). Nos dias de hoje, a computação gráfica está presente na maioria das áreas de conhecimento humano (MANSSOUR; COHEN, 2006), desde a engenharia que utiliza ferramentas CAD(*Computer-Aided Design*) para criação de projetos, ou a medicina que utiliza técnicas modernas de visualização de imagens para auxiliar em diagnósticos, ou então empresas que desenvolvem ferramentas capazes de criar animações para jogos eletrônicos.

Hoje em dia existem várias bibliotecas gráficas, logo a programação de aplicações em computação gráfica está mais simples (MANSSOUR; COHEN, 2006). Por exemplo a biblioteca *OpenGL* (*Open Graphics Library*), que será usada neste trabalho para criação dos objetos 3D, é uma biblioteca de rotinas gráficas portável e rápida, com modelagem 3D e 2D. Porém, antes de surgir essas bibliotecas gráficas ou antes até mesmo pensar em desenvolver aplicações gráficas, precisou-se considerar a evolução dos *hardwares* gráficos. Antigamente os computadores e impressoras só eram capazes de representar uma listagem alfanumérica, e foi na década de 50 que houve um marco importante na areá de *hardwares* gráficos. Em 1945 o *Massachusetts Institute of Technology* (MIT) começou a desenvolver o primeiro computador com recursos gráficos, o *Whirlwind* (WWI), que tinha o objetivo de realizar simulações de alta qualidade de aeronaves em tempo real (KAUFMAN et al., 1993).

2.4.1 Relação entre computação gráfica e visão computacional

A computação gráfica e a visão computacional são áreas que possuem uma relação bem próxima. A computação gráfica se preocupa com a criação e manipulação de imagens. Já a visão computacional é voltada para a análise dessas imagens, extraindo características como, por exemplo, o reconhecimento de faces ou impressões digitais.

Outra área que está relacionada à computação gráfica é a do processamento de imagens(PI). Esta área está voltada à análise dos dados da imagem, de forma que consiga captar informações e com isso melhorar as características visuais da imagem. Para realizar uma tarefa completa de processamento de imagens, são necessárias algumas etapas, que em alguns casos podem se tornar custosas. Algumas dessas etapas são:

- **Pré-processamento** - Nesta etapa são eliminados ruídos da imagem que possam ter sido obtidos na captura da imagem (fotografia, *scanner*, etc). Também é feita uma melhora no contraste da imagem.
- **Segmentação** - Esta etapa consiste em reconhecer e separar a imagem em regiões distintas, a fim de obter uma independência dos objetos reconhecidos.
- **Extração de Atributos** - Após reconhecer objetos na imagem, essa etapa irá atuar extraindo características desses objetos, como textura, forma, tamanho, etc.
- **Reconhecimento e Interpretação** - Esta etapa é voltada para análise dos dados obtidos pelas etapas anteriores e então classificar os objetos.

3 TRABALHOS RELACIONADOS

Nesta seção será apresentado o processo de revisão bibliográfica realizado para se contextualizar ao tema deste trabalho. Primeiramente estabelecemos o objetivo da pesquisa e, como resultado, selecionamos algumas palavras chaves para formar expressões de busca mais precisas em relação ao tema deste trabalho.

Palavras-chave:

- *Generation*
- *Creation*
- 3D *models*
- *Floor plans*
- *Recognition*

Para selecionar os trabalhos com mais relevância, foi preciso estabelecer alguns critérios.

- Os trabalhos devem estar relacionados à computação gráfica e construções civis.
- Os trabalhos devem trabalhar com reconhecimento e construção de plantas baixa.
- Os trabalhos devem estar escritos na língua portuguesa ou inglesa.

Os resultados das buscas trouxeram muitos trabalhos. Porém, após analisar com base nos critérios definidos, foram selecionados 5 artigos, que foram lidos e estudados por completo.

3.1 Resultados da pesquisa

Nesta seção será apresentado o resultado da revisão bibliográfica, contendo informações detalhadas sobre os trabalhos selecionados.

O trabalho proposto por Camozzato et al. (2015) descreve um método para reconstruir internos de construções de forma automatizada a partir de uma planta baixa e requisitos de sala definidos pelo usuário, apresentando resultados de boa qualidade. Para isso, o algoritmo irá detectar contornos e aberturas da construção usando computação gráfica. No primeiro momento o algoritmo irá pré-processar a imagem usando um Filtro Bilateral proposto por Tomasi e Manduchi (1998), para suavizar ruídos e preservar as bordas. Para detectar as bordas, é usado o algoritmo *watershed*, que é um algoritmo usado em bacias hidrográficas com o objetivo de inundar a parte mais funda e assim detectar suas bordas. No trabalho de Camozzato, esse algoritmo foi usado para facilitar a

detecção das linhas externas da planta baixa da construção. Para detectar as aberturas (portas e janelas), é usado o filtro de Sobel(SOBEL, 1968), cujo resultado será utilizado para ”recortar” as bordas da imagem e trabalhar individualmente sobre cada parte. Para as partes onde existe abertura, o algoritmo realizará duas operações. Primeiro, qualquer marcador próximo à borda longitudinal é removido. Isso evita que os cantos das paredes sejam identificados como bordas das aberturas. Segundo, qualquer marcador que não esteja na borda é descartado. Os marcadores remanescentes tornam-se pontos, porém alguns são falsos positivos que devem ser removidos. O algoritmo divide toda a planta baixa em células, para facilitar a decisão do que cada sala será. Para definir o interior das salas o algoritmo analisa as características (aberturas do tipo portas e janelas) de cada célula que contenha uma abertura, assim o algoritmo poderá definir a sala a partir destas características e a célula terá a informação do que tipo de sala será (banheiro, cozinha, quarto, etc). A etapa de expansão da sala é executada quando as células que contenham bordas estão bem definidas. O algoritmo irá expandir as salas para o seu tamanho final respeitando algumas regras, como “todas as salas devem poder ser acessadas a partir da sala inicial”, mantendo assim a conectividade entre as salas. Outro problema foi a concorrência por espaços pelas células, para isso foram usados os requisitos de sala definido pelo usuário.

O trabalho proposto por Okorn et al. (2010) descreve um método automatizado para criar modelos precisos de plantas baixas de construções já existentes. O método recebe como entrada um *point cloud* (pontos em um sistema de coordenadas tridimensionais), obtidos a partir de *scanners* posicionados em vários locais da construção. O *point cloud* contém vários pontos que apresentam as paredes, chão e teto da estrutura. Porém, para criar a planta baixa, é necessário que existam apenas os pontos que representam as paredes. Então é usado um histograma de dados de altura para detectar os pontos que representam o piso e do teto e removê-los. Para então projetar o restante dos pontos em um plano 2D e usando a transformada de Hough, detectar as linhas que formam as paredes. A motivação de desenvolver este método é que criar projetos baseados em construções já existentes é uma tarefa comum entre os profissionais da área de construção civil. No entanto, muitas dessas construções podem já não ser as mesmas da planta baixa original, devido a renovações da estrutura que não foram documentadas por exemplo. Dessa forma muitas vezes é necessário extrair dados da construção manualmente mesmo que a planta baixa original esteja disponível.

O trabalho proposto por Ahmed et al. (2014) descreve um algoritmo que consultará em um repositório de plantas baixas a partir de um esboço. O trabalho foi desenvolvido levando em conta que, quando arquitetos estão trabalhando em novos projetos e surge alguma situação que precisa ser resolvida, eles usam outros projetos arquitetônicos semelhantes para resolver o problema. Dessa forma, o algoritmo irá ajudar na busca do

projeto que mais se assemelha ao projeto atual. Atualmente, as buscas são feitas por meio de anotações na planta baixa, porém muitas vezes essas anotações não são precisas, logo não servem para uma busca eficiente. Para isso, foi preciso extrair a estrutura semântica do projeto do arquiteto e fazer uma busca no repositório de projetos antigos a fim de achar o mais semelhante e então mostrar os resultados para o usuário. Para extrair com precisão a semântica da planta baixa foi necessária uma etapa de segmentação fina para os diferentes tipos de informações presentes nas plantas baixas (textos, paredes, símbolos, etc), pois a informação que não é necessária para uma etapa específica será apenas ruído e poderá levar a resultados incorretos. A primeira etapa é a separação de textos dos gráficos. Para isso, foram usados os métodos apresentados por Ahmed et al. (2011), incluindo algumas melhorias voltadas para a planta baixa. Existe um método que separa as paredes dos símbolos usando um algoritmo de separação de linhas espessas e finas. O método resultará em duas imagens, uma com linhas espessas, contendo as paredes, portas e janelas e outra com linhas finas, contendo os símbolos. Neste trabalho, Ahmed aprimorou o método criando paredes médias e assim gerando uma terceira imagem, onde as paredes externas são espessas, as internas são médias e as finas são os símbolos, tudo isso obtido a partir de erosões sequências seguidas de dilatações. Para fazer o reconhecimento de plantas baixas, foi proposto um conceito onde a semântica extraída dos esboços são representadas como gráficos onde os vértices possuem um tipo indicando que tipo de sala é, e as arestas indicam se os vértices estão ligados diretamente ou são adjacentes. O algoritmo de recuperação foi baseado em uma versão do algoritmo de Messmer e Bunke (1995), onde as linhas e colunas são organizadas em árvore. A cada nível da árvore o vetor de coluna de linha é usado para encontrar o próximo nó da árvore usando uma estrutura de índice, e assim chegando à planta baixa mais semelhante a do arquiteto.

O trabalho proposto por Llados, Lopez-Krahe e Marti (1997) consiste em um sistema capaz de reconhecer desenhos arquitetônicos desenhos a mão em um ambiente CAD. Para isso, o documento de entrada é digitalizado e vetorizado. O documento de entrada é representado usando um gráfico atribuído de dois níveis, chamado de 2LG. No primeiro nível, a informação de linhas da imagem é representada por um conjunto de gráficos direcionados atribuídos, um para cada componente conectado. Os nós de cada gráfico representa os pontos característicos e os atributos representam a posição, grau e ângulo entre as linhas. O gráfico de segundo nível, é um hiper grafo, onde seus nós são gráficos de primeiro nível. Após isso, é utilizado técnicas de isomorfismo subgrafo para verificar a correspondência do gráfico de entrada e o gráfico de modelo. O problema do isomorfismo sub gráfico é equivalente ao problema de rotulagem consistente (Henderson 1990), que neste trabalho consiste em rotular os nós do modelo baseados nos nós de entrada. Dessa forma o gráfico pode ser representado como um gráfico de restrições, baseado nas arestas dos gráficos que são restrições geométricas e topológicas a serem

pelos nós em suas junções. A detecção de paredes será realizado pela detecção de áreas texturizadas. Porém por ser um modelo desenhado à mão, existem imperfeições nas retas. Então a imagem de entrada passa por um processo de vetorização e então as paredes são reconstruídas para o modelo final. Após isso, as paredes do gráfico de entrada são removidas do gráfico de entrada, e uma correspondência entre o restante do gráfico de entrada e o conjunto de gráfico do modelo é executada, a fins de encontradas a posição, escala e orientação de cada instância. Dessa forma o documento final pode ser construído a partir do gráfico com as paredes já definidas, instanciando cada modelo de acordo com seus atributos. Porém um pequeno problema de sobreposição acontece durante esse processo, mas é facilmente resolvido calculando a área sobreposta entre os elementos e reduzindo até que não haja mais sobreposição de nenhuma instância.

O trabalho apresentado por Merrell, Schkufza e Koltun (2010) consiste em uma ferramenta para geração automatizada de layouts de residências. Esses layouts são voltados na questão de organização do interior das residências. O programa recebe como entrada um conjunto de especificações da planta baixa (metragem quadrada aproximada, nº quartos, nº banheiros, nº janelas, etc) que muitas vezes estão incompletas. Porém o programa completa os requisitos de entrada com dados do mundo real, e assim retorna para o usuário uma planta baixa e um modelo 3D da residência com uma configuração realista. Um dos grandes desafios do trabalho foi organizar os cômodos da residência com base nos requisitos de entrada, pois em uma abordagem natural seria simplesmente calcular a probabilidade dos cômodos e seus adjacentes, como por exemplo, a cozinha geralmente fica adjacente à sala, ou então uma casa com 3 ou mais quartos geralmente possui uma sala de jantar separada. Porém essa abordagem não leva em consideração dependências adicionais entre várias salas adjacentes, que pode levar a projetos surreais. Para resolver esse problema, foi utilizado rede bayesiana treinada com dados do mundo real, que irá representar compactamente uma distribuição de probabilidade sobre um conjunto de informação de residências reais. Outro problema é a diferença na configuração de casas de número de andares diferentes, que embora seja possível treinar a rede, o número de dados seria muito elevado. O problema foi resolvido treinando 3 redes bayesianas separadas para residências de 1, 2 e 3 andares.

3.2 Análise dos trabalhos relacionados

Após a análise dos trabalhos relacionados, foi atribuído um grau de relevância em relação ao presente trabalho. O objetivo dessa avaliação é extrair as principais características dos trabalhos relacionados e identificar seus pontos fracos, para então aplicar-lós no presente trabalho.

Tabela 1 – Avaliação dos trabalhos.

Trabalhos	**Técnicas utilizadas**	**Relevância**
Procedural floor plan generation from building sketches	Reconhecimento de reta na imagem da planta baixa e geração da planta baixa em 3D	Alta
Automatic analysis and sketch-based retrieval of architectural floor plans	Reconhecimento de reta na imagem da planta baixa	Media
A system to understand hand-drawn floor plans using subgraph isomorphism and Hough transform	Reconhecimento de reta na imagem da planta baixa e geração de modelo 2D da planta baixa	Alta
Computer-Generated Residential Building Layouts	Geração de modelo 2D da planta baixa	Baixa
Toward Automated Modeling of Floor Plans	Reconhecimento de retas a partir de point cloud obtidos de scanners	Baixa

Fonte: o autor

O primeiro trabalho apresenta um método semelhante ao presente trabalho, fazendo um reconhecimento de uma planta baixa e criando uma representação 3D da construção. Entretanto, utiliza técnicas diferentes das que serão utilizadas no presente trabalho. Outra característica que o diferencia do trabalho proposto, é a especificação dos cômodos da construção. No trabalho de Camozzato et al. (2015), os cômodos são definidos por uma textura, já no presente trabalho haverá um recurso para organização de diferentes layouts de móveis nos cômodos para fazer essa representação.

O segundo trabalho possui uma relevância média, já que utiliza técnicas de reconhecimento de retas semelhantes às que serão utilizadas no presente trabalho. Outra característica é que a ferramenta recebe como entrada uma planta baixa desenhada à mão, necessitando tarefas como eliminar ruídos e correção de retas levemente sinuosas. Porém o trabalho não cria um modelo 3D para o usuário final, apenas apresentando plantas baixas de um repositório semelhantes à planta baixa de entrada.

O terceiro trabalho está voltado à captura de características de uma planta baixa para então exportá la para um ambiente CAD. A ferramenta recebe como entrada uma imagem digitalizada da planta baixa, e a partir disso é feita uma vetorização da imagem juntamente com correções nas imperfeições da planta, deixando-a perfeitamente alinhada em um ambiente CAD.

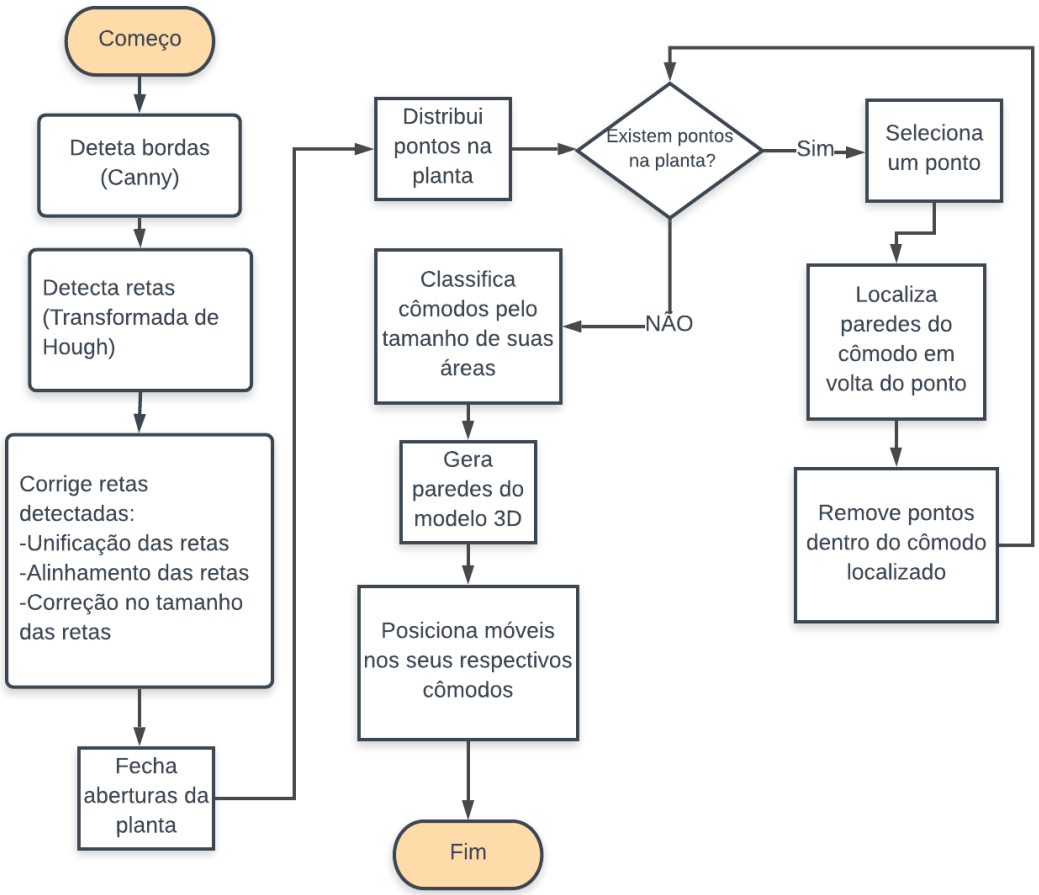
O quarto trabalho possui uma relevância baixa, pois não recebe como entrada uma imagem, e sim uma lista de requisitos. A ferramenta irá buscar, em um repositório, plantas baixas que atendam aos requisitos de entrada, e então criar um modelo 2D da planta baixa.

O quinto trabalho possui baixa relevância por alguns fatores, como não fazer um reconhecimento de características da planta baixa a partir de uma imagem e sim utilizando *scanners*. No entanto o trabalho usa um reconhecimento de retas em um *point cloud*, semelhante ao que será utilizado no presente trabalho.

4 FERRAMENTAS E MÉTODOS

A ferramenta foi desenvolvida utilizando a linguagem de programação *Python* para implementar todas as suas funcionalidades. Em cada uma das etapas, foram realizados testes para a validação de cada funcionalidade, para então prosseguir para a próxima etapa, objetivando dessa forma a redução do número de erros nas detecções. Podemos observar na Figura 3 o fluxograma que mostra uma visão geral das etapas da geração do modelo 3D desde filtragem de bordas até o posicionamento dos móveis em seus respectivos cômodos.

Figura 3 – Fluxograma das etapas da ferramenta.



Fonte: o autor

Para a validação dos algoritmos, incluindo desde a captura da imagem até os métodos de identificação e correção de retas, geração de paredes 3D e posicionamento dos móveis, coletamos 20 esboços de plantas baixas desenhados à mão por indivíduos escolhidos aleatoriamente, incluindo crianças de 12 a 16 anos durante a participação da universidade na 40ª Feira do Livro[1]. Além dessas, também foram incluídos no conjunto

[1] A 40ª Feira do livro aconteceu nos dias 11 a 16 de junho de 2019 na cidade de Alegrete-RS.

de imagens alguns esboços de plantas desenhadas por alunos de graduação da Unipampa.

4.1 Reconhecimento de retas

Nesta seção será apresentado o processo de reconhecimento das retas na imagem da planta baixa, as quais representam as paredes da construção na planta baixa. Para fazer esse reconhecimento, foi utilizada o método da transformada de Hough, a qual foi empregada em quatro dos cinco trabalhos descritos na seção 3.1 que faziam esse tipo de análise em imagens. Foi usado a implementação da transformada de Hough na biblioteca *OpenCV* (*Open Source Computer Vision Library*).

A OpenCV é uma biblioteca multiplataforma, completamente livre ao uso acadêmico e comercial, que auxilia no desenvolvimento de aplicativos de visão computacional. Esta biblioteca foi desenvolvida pela Intel no ano de 2000 e possui módulos de processamento de imagens, estrutura de dados, álgebra linear, GUI básica, entre outros (OPENCV, 2000).

4.1.1 Transformada de Hough

Na biblioteca OpenCV existem duas implementações da transformada de Hough, o método padrão, que basicamente retorna as retas da imagem assim como foi explicado na seção 2.3. O método utilizado neste trabalho é o transformada de Hough probabilística, que é uma implementação mais eficiente que a técnica de detecção de retas padrão do transformada de Hough, pois tem como saída os extremos das retas.

Após carregar a imagem, foi necessário fazer um pré-processamento de detecção de bordas. Para isso foi utilizado o filtro de Canny (CANNY, 1986), que detecta bordas a partir de critérios de quantificação de desempenho de operadores de bordas conhecidos como: critério de detecção e critério de localização.

Depois de realizada a detecção de bordas, foi aplicada a técnica de Hough, que recebeu como parâmetros: a saída do método de detecção de bordas, parâmetro ρ em *pixels*, parâmetro θ em graus, o número mínimo de intersecções para detectar uma reta, o número mínimo de pontos que podem formar uma reta e a distância máxima entre dois pontos que podem ser considerados pertencentes a mesma reta. O método retorna um conjunto de tuplas(ρ, θ). Os valores de ρ_i e θ_i são o tamanho da reta e o ângulo em relação ao eixo x, como já foi dito anteriormente neste trabalho. Para converter em pontos na forma (x_i, y_i), foram utilizadas as seguintes fórmulas:

$$x_i = \rho_i \cos \theta_i, \tag{4.1}$$

$$y_i = \rho_i \sin \theta_i. \tag{4.2}$$

Dessa forma conseguimos extrair os pontos que formam as retas da planta baixa.

4.1.2 Correção de retas detectadas

Antes de partir para a etapa de criação do modelo 3D, é necessário corrigir alguns problemas criados pelos algoritmos de Canny e pela transformada de Hough.

4.1.2.1 Retas paralelas

O filtro de Canny irá destacar as bordas da imagem, porém, cada uma das retas da planta baixa possuem uma certa espessura, logo, o filtro de Canny irá detectar duas bordas para cada reta, como podemos observar na Figura 4. Onde as retas em preto representam as bordas da reta em vermelho. Dessa forma, cada uma das retas irá possuir duas bordas.

Figura 4 – Bordas de uma reta.

Fonte: o autor

Quando a transformada de Hough realiza a detecção de retas na imagem resultante da detecção de bordas do algoritmo Canny, serão detectadas as duas bordas de cada reta. Isso irá criar um problema de retas paralelas. Outro problema gerado pela transformada de Hough é a questão de que esse algoritmo detecta somente retas perfeitamente retas, o que será pouco provável em uma imagem de uma planta desenhada a mão. Podemos observar na Figura 5 que existem vários segmentos de retas(em vermelho) que pertencem ao segmento de reta A (em preto).

Figura 5 – Segmentos de retas.

Fonte: o autor

Para resolver esses dois problemas foi desenvolvido um método de unificação de retas representado no pseudo código da Código Fonte 4.1.

Código Fonte 4.1 – Unifica retas Fonte: o autor

```
1  //dist = maximum distance between the lines
2  //lines = set of all lines obtained by the unification of lines
3  INPUT dist, lines
4  FOR  i = 0 TO ((n lines)-1) {
5      R1 = lines[i]
6      FOR j = i + 1 TO ((n lines)-1) {
7          R2 = lines[j]
8          //Checks whether the lines analyzed are the same
9          IF R1 != R2 THEN {
10             //Both straight lines are vertical
11             IS R1.angle == 90 AND R2.angle == 90{
12                 //Compares the distance of the x-axis lines
13                 IF |(R1.x1 - R2.x1)| < dist AND |(R1.x2 - R2.x2)| < dist {
14                     //Values of y of straight line R1 is modified to have
15                     //the length maximum  between the two lines R1 and R2
16                     R1.y1 = MINIMUM_VALUE_BETWEEN R1.y1 AND R2.y1
17                     R1.y2 = MAXIMUM_VALUE_BETWEEN R1.y2 AND R2.y2
18                 }
19             }
20             //Both straight lines are horizontal
21             ELSE IF R1.angle == 0 AND R2.angle == 0 {
22                 //Compares the distance of the y-axis lines
23                 IF |(R1.y1 - R2.y1)| < dist AND |(R1.y2 - R2.y2)| < dist {
24                     //Values of x of straight line R1 is modified to have
25                     //the length maximum  between the two lines R1 and R2
26                     R1.x1 = MINIMUM_VALUE_BETWEEN R1.x1 AND R2.x1
27                     R1.x2 = MAXIMUM_VALUE_BETWEEN R1.x2 AND R2.x2
28                 }
29             }
30         }
31     }
32 }
33
34     //Removes the line r2 from the set of lines
35     REMOVE R2 OF lines
```

Este método irá usar dois laços de repetição para percorrer o conjuntos de retas. No primeiro laço é selecionado uma reta do conjunto de retas chamada R1, para então

ser comparada com as demais. O segundo laço selecionará a primeira reta a ser analisada com a $R1$, chamada de $R2$. Após selecionada duas retas para análise, o algoritmo irá testá-las para verificar se não são as mesmas retas, caso isso aconteça, o segundo laço irá selecionar outra $R2$ e verificar novamente. Quando as retas $R1$ e $R2$ forem diferentes, o algoritmo irá verificar o ângulo das retas, a fim de saber se ambas as retas estão no mesmo sentido (vertical ou horizontal). Se as retas estiverem em sentidos diferentes, o segundo laço irá selecionar outra $R2$ e repetir todo o processo. Caso as retas estejam no mesmo sentido, existem dois caminhos semelhantes para o algoritmo seguir, o caminho das retas verticais (linha 11) e o caminho das retas horizontais (linha 20). Se ambas as retas estiverem na horizontal, o algoritmo irá calcular a distância entre as retas no eixo x para saber se as retas estão próxima, e também irá verificar se as retas satisfazem as seguintes condições (linha 13 para verticais e 22 para horizontais):

- Para retas horizontais:

$$R1.x_1 \geq R2.x_1, \tag{4.3}$$

$$R1.x_2 \leq R2.x_2. \tag{4.4}$$

- Para retas verticais:

$$R1.y_1 \geq R2.y_1, \tag{4.5}$$

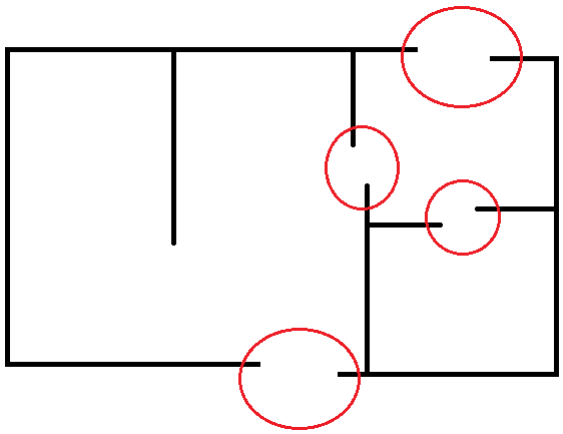
$$R1.y_2 \leq R2.y_2. \tag{4.6}$$

Se essas condições forem satisfeitas, o algoritmo vai alterar os valores de x_1 e x_2 da reta $R1$ caso as retas forem horizontais e y_1 e y_2 caso forem verticais. Os novos valores da reta $R1$ serão o menor valor entre $R1.x_1$ e $R2.x_1$ para $R1.x_1$ (linha 26), e o maior valor entre $R1.x_2$ e $R2.x_2$ para $R1.x_2$ (linha 27). Caso as retas estejam na vertical o processo será o mesmo, só que os valores usados serão em relação ao eixo y (linhas 16 e 17). Dessa a forma a reta $R1$ terá o tamanho maior tamanho entre as duas retas. Após modificar os valores da reta $R1$, o algoritmo removerá a reta $R2$ do conjunto de retas (linha 29).

4.1.2.2 Retas não alinhadas

Após os métodos de unificação das retas e a correção dos comprimentos das mesmas, ainda é preciso corrigir outro problema que é o desalinhamento de algumas retas, como podemos observar na Figura 6.

Figura 6 – Retas não alinhadas.



Fonte: o autor

Para resolver esse problema foi criado um método para alinhar retas representado no pseudocódigo da Código Fonte 4.2.

Código Fonte 4.2 – Pseudo código do método de alinhamento de retas. Fonte: o autor

```
1  //dist= maximum distance tolerated between lines
2  //lines = set of all lines obtained by the Hough transform
3  INPUT dist, list
4  FOR i = 0 TO ((n lines)-1) {
5      R1 = lines[i]
6      FOR j = i + 1 TO ((n lines)-1) {
7          R2 = lines[j]
8          //Checks if R1 and R2 are the same line
9          IF R1 != R2 {
10             //Both R1 and R2 lines are vertical
11             IF R1.angulo == 90 AND R2.angulo ==90 {
12                 //Compares distances of R1 and R2 in axis x
13                 IF |(R1.x1 - R2.x1)| < dist {
14                     //y values of R2 are modified for alignment with R1
15                     R2.x1 = R1.x1
16                     R2.x2 = R1.x2
17                 }
18             }
19             //Both R1 and R2 are horizontal
20             ELSE IF R1.angle == 0 AND R2.angle == 0 {
21                 //Compares distances of R1 and R2 in axis y
22                 IF |(R1.y1 - R2.y1)|< dist {
23                     //x values of R2 are modified for alignment with R1
24                     R2.y1 = R1.y1
25                     R2.y2 = R1.y2
26                 }
27             }
28         }
29     }
30 }
```

Esse método também possui dois laços de repetição para comparar todos os pares de retas. Também será realizado processo semelhante para verificar se as duas retas $R1$ e $R2$ não são a mesma e também se os ângulos das duas retas são iguais. Além destas duas condições, ainda é preciso calcular a distância das retas no eixo x (para retas verticais) ou eixo y (para retas horizontais). Caso as retas estejam próximas, o algoritmo irá modificar os valores da reta $R2$ da seguinte maneira:

- Para retas horizontais:

$$R2.y_1 = R1.y_1, \tag{4.7}$$

$$R2.y_2 = R1.y_2. \tag{4.8}$$

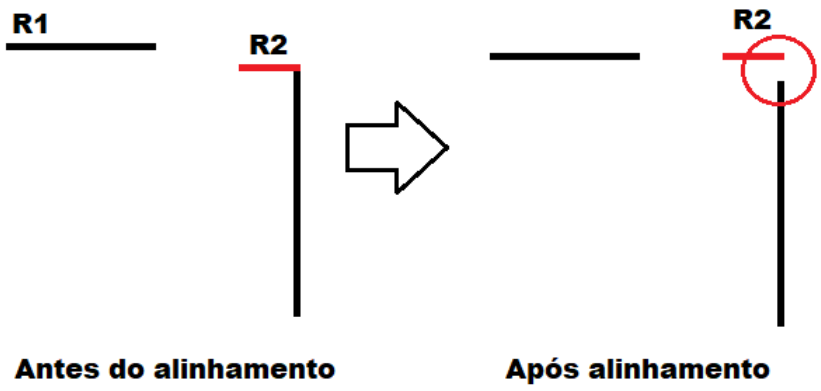
- Para retas verticais:

$$R2.x_1 = R1.x_1, \tag{4.9}$$

$$R2.x_2 = R1.x_2. \tag{4.10}$$

Dessa forma, todas as retas ficarão alinhadas. Porém, isso irá criar aberturas na planta, já que as retas serão movidas. Observe a imagem Figura 7, onde o alinhamento das retas foi corrigido. Podemos ver que o alinhamento dessas duas retas, criou uma abertura quando moveu a reta $R2$ para cima em direção a reta $R1$. Para resolver esse novo problema, foi criado outro método para corrigir o tamanho das retas.

Figura 7 – Problema gerado pelo alinhamento de retas.



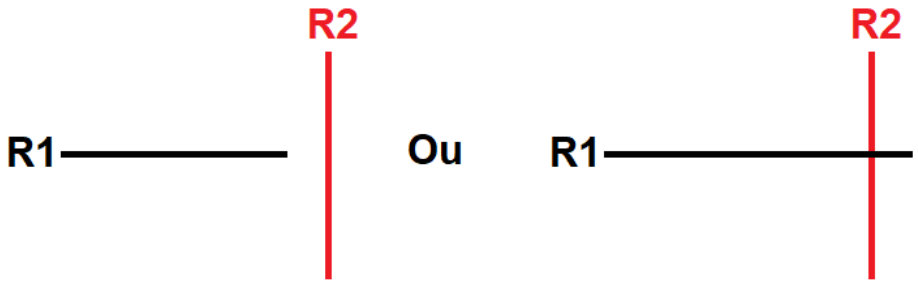
Fonte: o autor

4.1.2.3 Abertura entre retas

Como explicado anteriormente, o método de alinhamento acaba gerando aberturas nas retas que representam a planta baixa, ou então, acabam por deixar uma reta maior

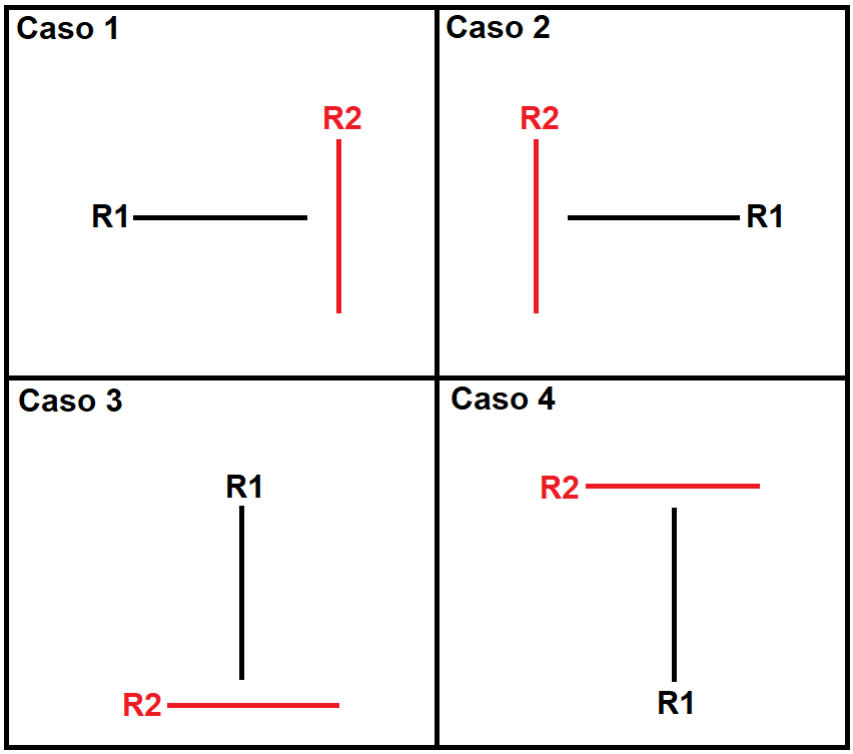
do que deveria, podemos observar exemplo dos dois problemas na Figura 8. Para resolver esse problema foi desenvolvido e implementado um método. O algoritmo irá analisar as quatro possíveis posições de uma reta $R1$ em relação a uma outra reta $R2$, como podemos observar na Figura 9.

Figura 8 – Casos de problema no tamanho das retas.



Fonte: o autor

Figura 9 – Possíveis posições de $R1$ em relação a $R2$.



Fonte: o autor

O método de correção dos tamanhos das retas irá funcionar de maneira similar aos outros dois métodos apresentados anteriormente, por meio do uso de dois laços aninhados

para percorrer o conjuntos de pares de retas e compará-las, como podemos observar no Código Fonte 4.3.

Código Fonte 4.3 – Pseudo código do método de correção no tamanho das retas. Fonte: o autor

```
1  //dist =  maximum distance between points on straight lines
2  //retas = set of all lines obtained by the straight line alignment
3  INPUT dist, retas
4  FOR i = 0 TO ((n retas)-1) {
5      R1 = retas[i]
6      FOR j = i + 1 TO ((n retas)-1) {
7          R2 = retas[j]
8          //Checks if R1 and R2 are the same line
9          IF R1 != R2{
10             //R1 vertical
11             //R2 horizontal
12             IF R1.angulo == 90 E R2.angulo == 0 {
13                 //Calculates the distance between the end point y of the line R1
                       and the point y of the line R2
14                 IF |(R1.y2 - R2.y2)| < dist {
15                     //Caso 3
16                     R1.y2 = R2.y2
17                 }
18                 //Calculates the distance between the initial point y of the line
                       R1 and the point y of the line R2
19                 ELSE IF |(R1.y1 - R2.y1)| < dist {
20                     //Caso 4
21                     R1.y1 = R2.y1
22                 }
23             }
24             //R1 horizontal
25             //R2 vertical
26             ELSE IF R1.angulo == 0 E R2.angulo == 90 {
27                 //Calculates the distance between the end point x of the line R1
                       and the point x of the line R2
28                 IF |(R1.x2 - R2.x2)| < dist {
29                     //Case 1
30                     R1.x2 = R2.x2
31                 }
32                 //Calculates the distance between the initial point x of the line
                       R1 and the point x of the line R2
33                 ELSE IF |(R1.x1 - R2.x1)| < dist {
34                     //Case 2
35                     R1.x1 = R2.x1
36                 }
37             }
38         }
39     }
40 }
```

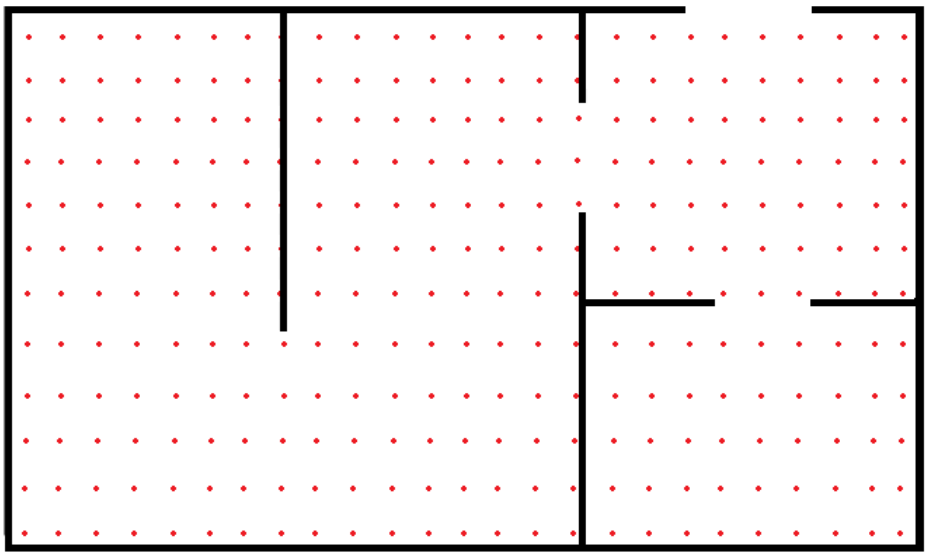
A diferença é que os ângulos das retas $R1$ e $R2$ agora devem ser diferentes. Para saber em qual dos quatro casos possíveis as retas $R1$ e $R2$ se encontram, foi criados alguns testes com base na distância entre pontos das retas. Quando a reta $R1$ é horizontal e reta $R2$ é vertical, é levado em consideração somente a proximidade das retas no eixo x.

Quando a reta $R1$ for vertical e a reta $R2$ for horizontal, será levado em consideração somente a proximidades das retas no eixo y. Todos os quatro testes e as respectivas soluções são muito semelhantes. Sabendo isso, usaremos para a explicação apenas o caso 1 (linha 28 à linha 30), que é acionado quando os pontos $R1.x_2$ e $R2.x_2$ estão próximos (linha 28). Se essa condição for satisfeita, o algoritmo irá modificar os valores da reta $R1$, para que o $R.1x_2$ receba o valor de $R2.x_2$ (linha 30).

4.2 Classificação dos cômodos

A partir das retas obtidas nas etapas de detecção e correção de retas, é realizado a localização dos cômodos. O algoritmo irá usar pontos distribuidor no interior de toda a planta baixa para detectar as paredes de cada cômodo, podemos observar um exemplo dos pontos distribuídos em uma planta baixa na Figura 10.

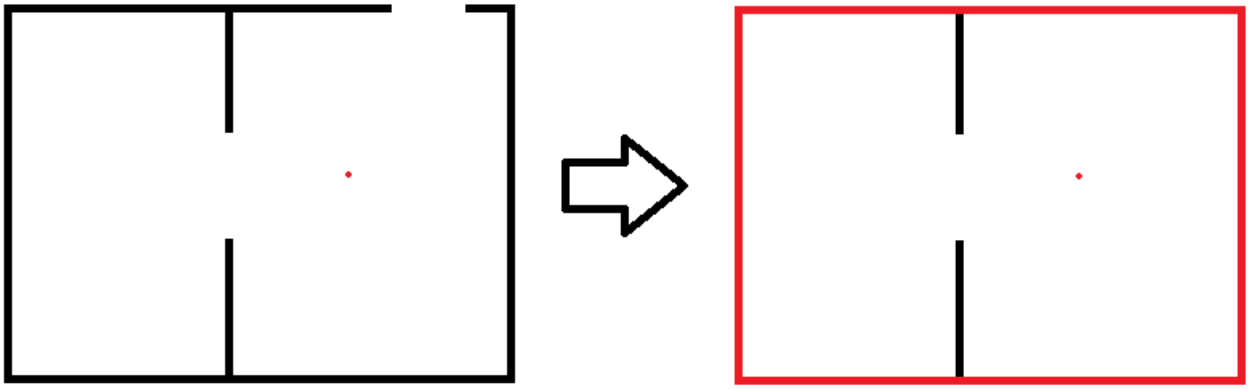
Figura 10 – Distribuição de pontos na planta baixa.



Fonte: o autor

Porém, antes de fazer essa localização, é necessário fechar as aberturas dos cômodos pois, sem isso, alguns cômodos podem ser localizados de forma incorreta, como podemos observar na Figura 11.

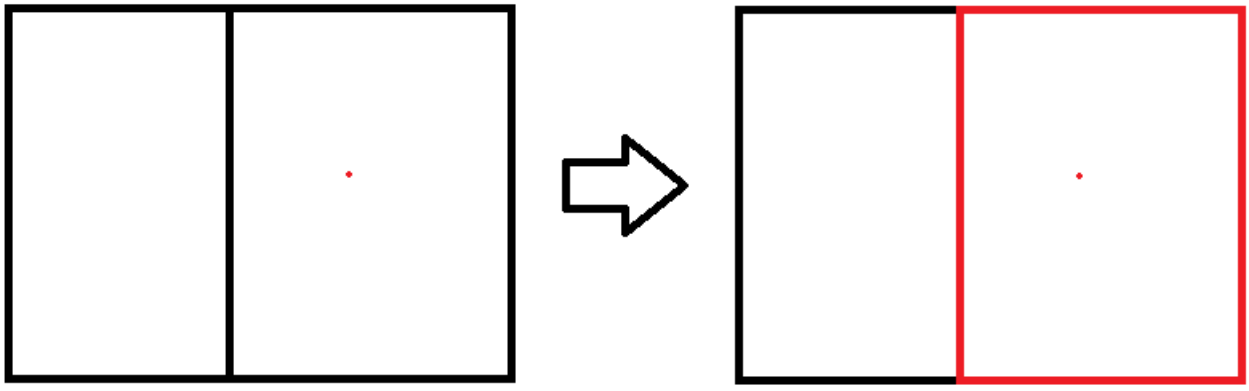
Figura 11 – Identificação errada do cômodo.



Fonte: o autor

Para corrigir esse problema, antes de aplicar o método para localizar os cômodos, o algoritmo procura as aberturas da planta. Para localizar as aberturas, o algoritmo irá identificar retas que possuem o mesmo ângulo e que estão alinhadas. Identificando essas retas, o algoritmo irá remover uma das retas, e modificar o comprimento da outra para que tenha o tamanho máximo entre as duas. Dessa forma, o problema de localização dos cômodos é resolvido, como podemos ver na Figura 12.

Figura 12 – Identificação certa do cômodo



Fonte: o autor

Podemos observar a seguir o pseudocódigo do método de localização dos cômodos no Código Fonte 4.4. Porém, usaremos para a explicação somente o trecho onde o algoritmo localiza a parede da direita de um cômodo (linha 19 à 27).

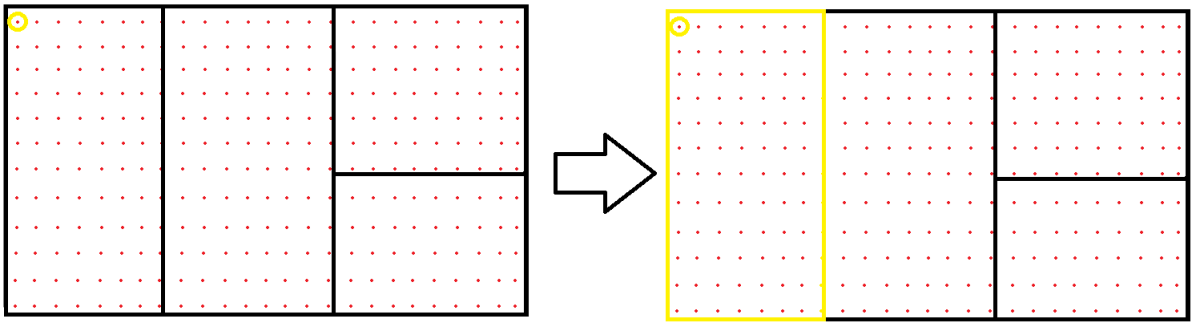
Código Fonte 4.4 – Pseudo código do método de localização de cômodos. Fonte: o autor

```
//retas = set of all lines
//points = set of all points
INPUT points, retas
FOR i = 0 TO ((n points)-1) {
    point = points[i]
    //Initial walls of a room with their respective initial distances
    //First value of the tuple: straight representing the wall
    //Second value of the tuple: distance from the point to the line
    Pright = [Nulo, INF]
    Pleft = [Nulo, INF]
    Ptop = [Nulo, INF]
    Pbottom = [Nulo, INF]
    FOR j = 0 TO ((n retas)-1) {
        R1 = retas[j]
        //Calculate distance between point and line R1 using Pythagorean theorem
        distance = calculate_distance(R1, point)
        //R1 vertical
        IF (R1.angulo == 90) {
            //Wall R1 is to the right of the point
            IF (point.x < R1.x) {
                //If the distance is less than the distance from the
                //previous distance (initially set to a very high value)
                IF (distance < Pright[1]) {
                    //First value of the tuple: wall to the right of the point is
                        defined in R1
                    Pright[0] = R1
                    //Second value of the tuple: Previous distance receives the
                        new value
                    Pright[1] = distance
                }
            }
            //Wall R1 is to the left of the point
            ELSE IF (point.x > R1.x) {
                //If the distance is less than the distance from the
                //previous distance (initially set to a very high value)
                IF (distance < Pleft[1]) {
                    //First value of the tuple: wall to the left of the point is
                        defined in R1
                    Pleft[0] = R1
                    //Second value of the tuple: Previous distance receives the
                        new value
                    Pleft[1] = distance
                }
            }
        }
        //R1 horizontal
        ELSE IF (R1.angulo == 0) {
            //Wall R1 is above the point
            IF (point.y > R1.y) {
                //If the distance is less than the distance from the
                //previous distance (initially set to a very high value)
                IF (distance < Ptop[1]) {
                    //First value of the tuple: wall above the point is defined
                        in R1
                    Ptop[0] = R1
```

```
51                      //Second value of the tuple: Previous distance receives the
                            new value
52                      Ptop[1] = distance
53                  }
54              }
55              //Wall R1 is below the point
56              ELSE IF (point.y < R1.y) {
57                  //If the distance is less than the distance from the
58                  //previous distance (initially set to a very high value)
59                  IF (distance < Pbottom[1]) {
60                      //First value of the tuple: wall below the point is defined
                            in R1
61                      Pbottom[0] = R1
62                      //Second value of the tuple: Previous distance receives the
                            new value
63                      Pbottom[1] = distance
64                  }
65              }
66          }
67      }
68      //Remove all points that are inside the room
69      REMOVE_POINT(Pright , Pleft, Pbottom, Ptop)
70  }
```

Com os pontos distribuídos na planta com as aberturas fechadas, o algoritmo irá funcionar selecionando o primeiro ponto do conjunto de pontos. A partir desse ponto, o algoritmo fará uma busca procurando as retas acima, abaixo, à direita e à esquerda. O método possui variáveis em forma de tuplas para as paredes ($P_{direita}$, $P_{esquerda}$, P_{cima}, P_{baixo}), onde o primeiro valor da tupla representa a reta de cada uma das paredes, inicializadas em um valor nulo (linha 9 à linha 12), e o segundo valor da tupla representa a distância do ponto até e parede, inicializadas em um valor muito alto (infinito). O método possui dois laços aninhados, o laço externo irá percorrer o conjunto de pontos (linha 4) e o laço interno irá percorrer o conjunto de retas (linha 13). Como dito antes, será selecionado o primeiro ponto do conjunto de pontos (linha 5) e irá calcular a distância entre o ponto e a reta $R1$ selecionada (linha 16) usando o teorema de Pitágoras. Após isso, o algoritmo irá procurar uma reta vertical (linha 18) que esteja à direita (linha 20) do ponto (nesse caso onde estamos procurando a parede à direita do ponto). Quando encontrada uma reta a direita do ponto, o algoritmo irá verificar se a distância do ponto até a reta $R1$ (23) é menor que a distância da parede à direita do ponto (inicializada em um valor muito alto justamente para que a primeira parede encontrada à direita do ponto seja escolhida, as demais paredes serão escolhidas ou não apenas pela distância que se encontram do ponto), se a distância for menor que o segundo valor da tupla $Pdireita_1$, o algoritmo irá modificar o valor da $Pdireita_0$ para a reta $R1$ (linha 25) e modificar o valor da $Pdireita_1$ para a distância em que a reta se encontra do ponto. Após percorrer todo o laço interno (laço das retas), as variáveis das paredes terão formando um quadrilátero que representa aquele cômodo como podemos ver na Figura 13.

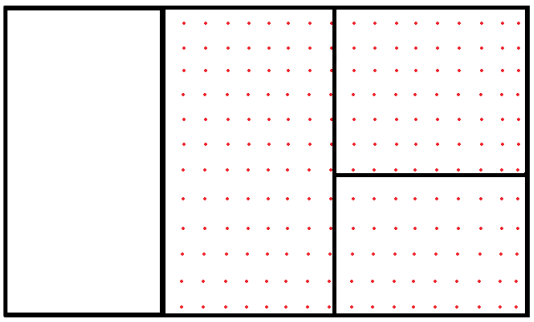
Figura 13 – Identificação de um cômodo da planta.



Fonte: o autor

Após localizar um cômodo, o algoritmo irá remover todos os pontos que estiverem dentro desse quadriláteros (linha 61), como podemos ver na Figura 14. Após isso o algoritmo irá escolher novamente um dos pontos que restaram de forma aleatória e repetir o procedimento até que não existam mais pontos.

Figura 14 – Remoção de pontos do cômodo identificado.



Fonte: o autor

Quando não houverem mais pontos na planta, todos os cômodos terão sido localizados e armazenados em um conjunto de cômodos. Após isso, o algoritmo irá classificar todos os cômodos com base na sua área. Essa classificação será feita conforme a Tabela 2.

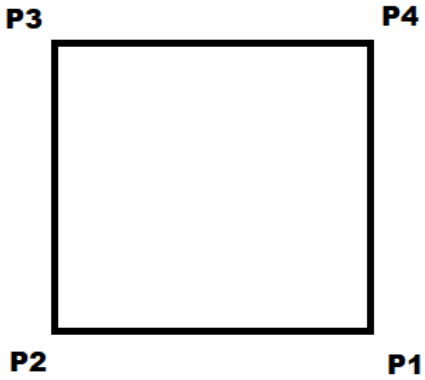
Tabela 2 – Classificação dos cômodos.

Cômodo	Área	Classificação
cômodo 1	menor	banheiro
cômodo 2	segundo menor	cozinha
cômodo 3	médio	quarto
cômodo 4	médio	quarto
cômodo n	maior	sala

4.3 Posicionamento de moveis dentro dos cômodos

Conforme apresentado na seção seção 4.2, os cômodos serão classificados pelo tamanho da sua área. Após a identificação dos cômodos, os móveis serão posicionados na seguinte ordem: móveis do banheiro, móveis da cozinha, móveis da sala e móveis do(s) quarto(s). O algoritmo sempre irá contar quantos cômodos existem na planta antes de começar a distribuir móveis, dando prioridade na ordem de posicionamento mencionada anteriormente. A posição dos móveis será baseada nos cantos de cada cômodo. Para facilitar o entendimento, utilizaremos a Figura 15 para guiar a explicação.

Figura 15 – Representação dos pontos de um cômodo



Fonte: o autor

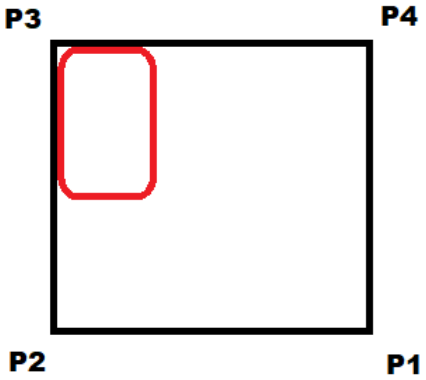
Para posicionar um objeto qualquer de tamanho 10x20 (largura X comprimento) com o sentido na vertical, o algoritmo irá calcular a posição do cômodo em:

$$x = (P3.x - (objeto.largura/2)) \tag{4.11}$$

$$y = (P3.y - (objeto.comprimento/2)) \tag{4.12}$$

Dessa forma, o objeto terá a distância suficiente da parede superior e a parede a esquerda, como podemos observar na Figura 16.

Figura 16 – Representação do objeto posicionado no canto superior esquerdo do cômodo.

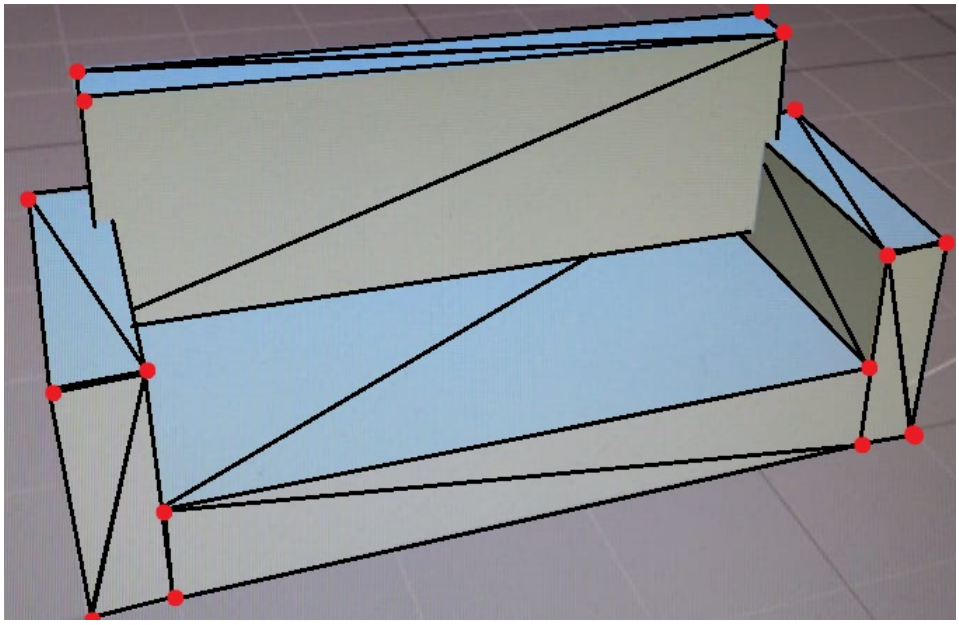


Fonte: o autor

4.3.1 Estrutura dos móveis

Os móveis utilizados foram criados através da plataforma web *Threejs* (CABELLO, 2010), que é uma estrutura *JavaScript* gratuita, utilizada para exibição, edição e criação de objetos 3D. Os móveis são estruturados com vários pontos que representam os vértices do móvel, esses pontos são interligados três a três, formando triângulos que irão representar as faces do móvel, como podemos observar na Figura 17.

Figura 17 – Representação da estrutura de um móvel.



Fonte: o autor

4.4 Construção do modelo 3D

Nesta etapa mostraremos as ferramentas e métodos criados para a criação das paredes em 3D a partir das retas encontradas na detecção e correção de retas. Por meio de pesquisas, decidimos usar a biblioteca *OpenGL* (*Open Graphics Library*) para a criação das paredes e também a criação dos objetos internos dos cômodos(móveis).

4.4.1 OpenGL

Atualmente existem bibliotecas gráficas disponíveis que facilitam a programação das aplicações. O *OpenGL* (*Open Graphics Library*), que é uma API(*Application Programming Interface*) livre, utilizada na computação gráfica para criação e manipulação de imagens bidimensionais (2D) e tridimensionais (3D) (OPENGL, 2018).

Além da criação de aplicações 2D e 3D, o *OpenGL* também dá suporte a iluminação, colorização, mapeamento de textura, animação, entre muitos outros tipos de efeitos. *OpenGL* é reconhecida e aceita como um padrão API para desenvolvimento de aplicações gráficas 3D em tempo real (MANSSOUR; COHEN, 2006).

Os objetos criados com *OpenGL* consistem em simples primitivas gráficas que podem ser combinadas de várias formas (WRIGHT; SWEET, 1999). As paredes da construção serão formadas por vértices. Com isso será necessário criar uma lista de vértices com os pontos das retas obtidos na seção 4.1.

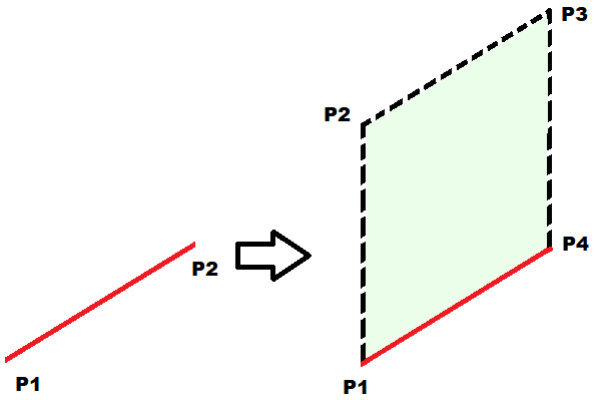
4.4.2 Geração das paredes e cômodos da planta 3D

As paredes da planta 3D serão geradas a partir das retas obtidas na detecção e correção de retas. O *OpenGL* cria faces a partir de pontos passados para o método. O algoritmo irá percorrer o conjunto de retas, para cada reta, o algoritmo irá gerar quatro pontos que formarão um polígono que representará uma parede. Cada polígono será formado da seguinte maneira:

- ponto 1 = ($R1.x_1$, $R1.y_1$, 0)
- ponto 2 = ($R1.x_1$, $R1.y_1$, 1)
- ponto 3 = ($R1.x_2$, $R1.y_2$, 1)
- ponto 4 = ($R1.x_2$, $R1.y_2$, 0)

O terceiro valor de cada ponto representa o eixo z do plano, os valores 1 e 0 representam a altura da planta. O algoritmo vai basicamente ligar os pontos: ponto 1 no ponto 2, ponto 2 no ponto 3, ponto 3 no ponto 4 e ponto 4 de volta no ponto 1, como mostra a Figura 18.

Figura 18 – Resultado da terceira etapa da detecção de retas

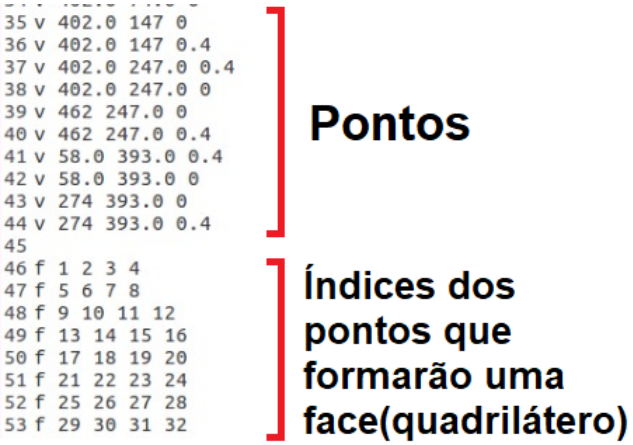


Fonte: o autor

4.4.3 Exportação planta baixa

Além da geração do modelo 3D da planta baixa, a aplicação também possibilita a exportação da planta baixa em arquivos no formato OBJ, que é um formato de arquivo que armazena malhas poligonais tridimensionais. A estrutura da planta baixa armazenada nesse arquivo é semelhante a estrutura dos móveis, descrita na subseção 4.3.1. A diferença é que as faces do objeto são quadriláteros, e não triângulos como são a estrutura dos móveis. Podemos observar um exemplo de arquivo OBJ de uma das plantas baixas usada nos testes na Figura 19

Figura 19 – Estrutura arquivo OBJ da planta baixa



Fonte: o autor

A exportação desse tipo de arquivo traz novas possibilidades do uso da ferramenta, por exemplo, pode-se usar a detecção e correção da linhas para extrair as posições da

paredes e exporta um arquivo OBJ que poderá ser importado em outras ferramentas, por exemplo a ferramenta AUTOCAD(INC., 1982), que é usada para desenhos técnicos de peças industriais.

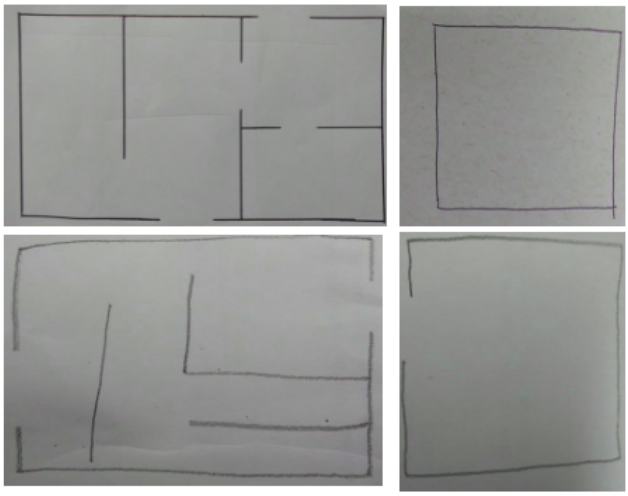
5 RESULTADOS

Neste capítulo serão apresentados os resultados obtidos ao longo do processo inteiro, cobrindo desde a detecção das retas até a criação do modelo 3D. Inicialmente, serão apresentados os esboços digitalizados a partir dos desenhos à mão livre no papel e, em seguida, todas as etapas incluindo a obtenção dos vértices das retas, as filtragens necessárias nas retas cruas obtidas, seguido pela geração do modelo tridimensional com a detecção dos cômodos, seu preenchimento automático com os móveis e, finalmente, a visualização em três dimensões.

5.1 Resultados detecção de retas

Para demonstração dos resultados da detecção e correção de retas, foram selecionadas 4 imagens para representar a evolução do reconhecimento e das três etapas do melhoramento das retas. A discussão dos resultados compreendendo todo o conjunto das 20 imagens utilizadas será feito na subseção 5.2.3. Podemos observar as plantas baixas desenhadas à mão na Figura 20.

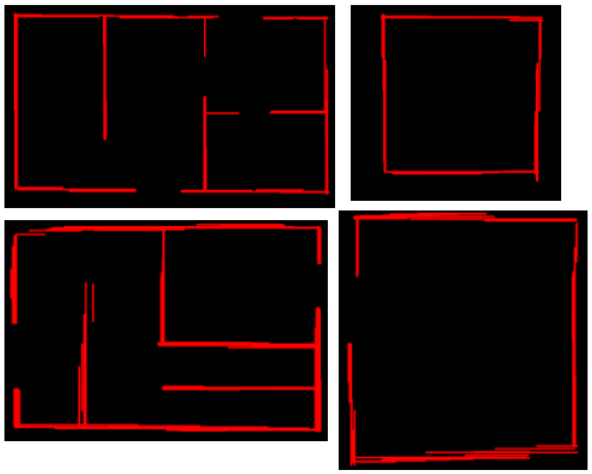
Figura 20 – Plantas baixas de entrada.



Fonte: o autor

Conforme apresentado anteriormente na metodologia, a detecção de retas via transformada de Hough implementada pela biblioteca *OpenCV* apresentava alguns problemas que precisavam ser resolvidos antes de passar para a próxima etapa. Na Figura 21 podemos observar o resultado obtido utilizando somente a transformada de Hough e alguns de seus problemas.

Figura 21 – Resultado da primeira etapa da detecção de retas.

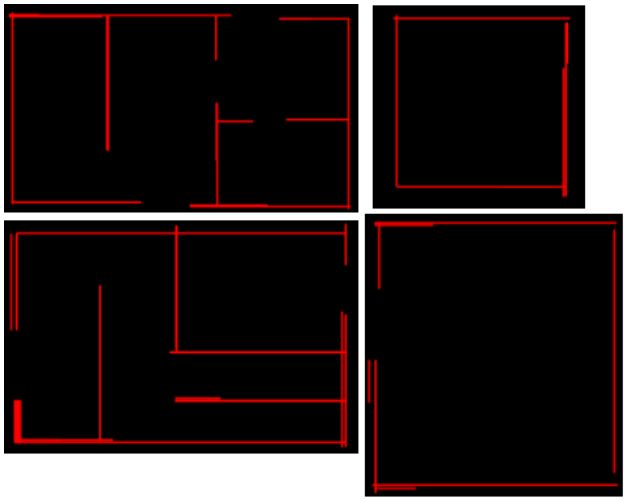


Fonte: o autor

5.1.1 Resultados da etapa 1 de melhoramento das retas

Depois da detecção realizada pela transformada de Hough, as retas obtidas ainda não se encontravam em um formato satisfatório, pois a transformada retornava múltiplas retas de curta extensão e sobrepostas onde deveria se encontrar, por exemplo, uma única parede. Por essa razão, foram submetidas ao método desenvolvido neste trabalho para a sua filtragem e unificação. O resultado após esse método é mostrado na Figura 22.

Figura 22 – Resultado da segunda etapa da detecção de retas.

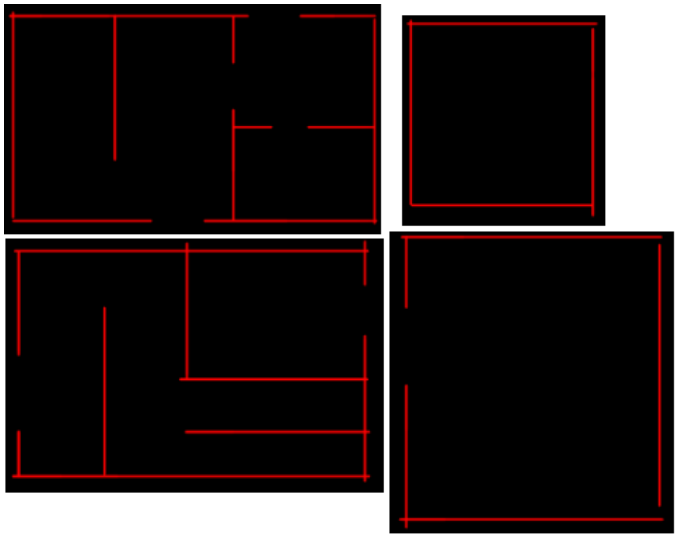


Fonte: o autor

5.1.2 Resultado da etapa 2 de melhoramento das retas

Após a etapa de filtragem anterior com remoção de segmentos curtos e unificação de retas próximas, tratou-se o problema de fragmentação e multiplicidade, mas as retas resultantes continuavam com problemas de alinhamento. Dessa forma, aplicou-se outro método desenvolvido neste trabalho, responsável pelo alinhamentos das retas resultantes da etapa anterior (principalmente onde existem aberturas) e produzir um resultado mais adequado. Podemos observar o resultado obtido após o uso desse método na Figura 23.

Figura 23 – Resultado da segunda etapa da detecção de retas.

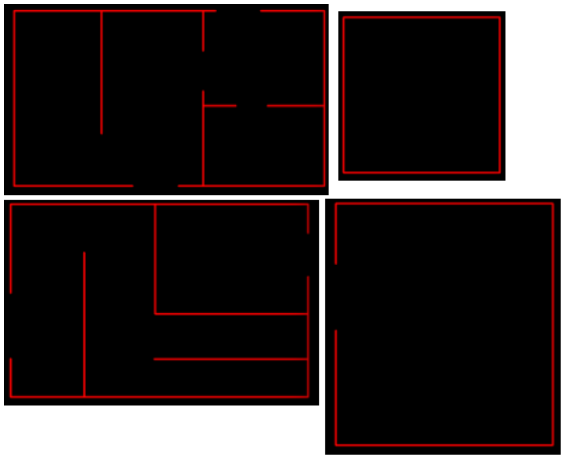


Fonte: o autor

5.1.3 Resultado da etapa 3 de melhoramento das retas

Após concluído o alinhamento das retas, é preciso compatibilizar os seus comprimentos pois a geração das quinas pode ser seriamente comprometida. Aplicamos então o algoritmo que irá resolver o problema do comprimento das retas, fazendo com que suas terminações coincidam. Podemos observar nas imagens que, após o alinhamento, algumas retas ainda se encontram com dimensões maiores ou menores que o tamanho correto. Para corrigir esse problema, foi utilizado o método desenvolvido para ajuste do tamanho das retas. O resultado pode ser observado na Figura 24.

Figura 24 – Resultado da terceira etapa da detecção de retas.

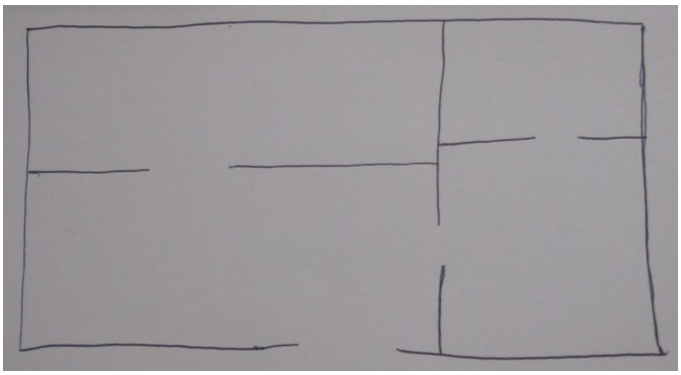


Fonte: o autor

5.2 Resultados criação do modelo 3D

Para a demonstração dos resultados envolvendo a criação do modelo 3D e posicionamento dos móveis nos cômodos, foram selecionados 2 esboços de plantas baixas do conjunto de imagens de teste. O critério de escolha dos 2 esboços de planta baixa foi a semelhança visual com plantas baixas reais. Podemos observar os esboços de plantas baixas na Figura 20 e Figura 25 (criada por crianças de 12 a 16 anos na 40ª Feira do Livro de Alegrete 2019).

Figura 25 – Planta baixa obtida na 40ª Feira do Livro de Alegrete 2019.



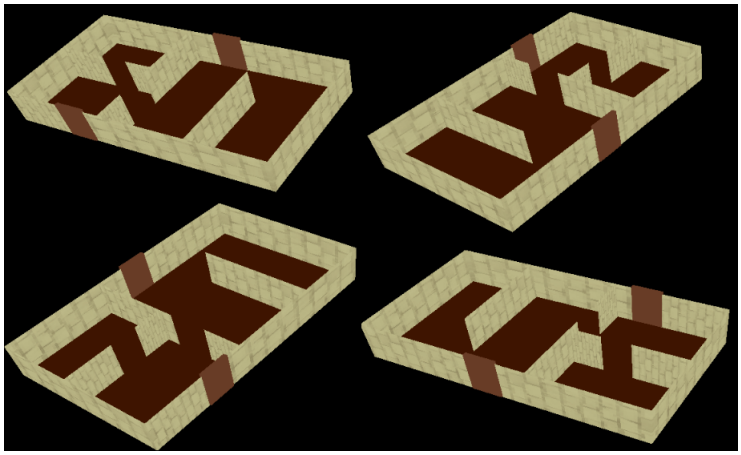
Fonte: o autor

5.2.1 Resultado criação das paredes do modelo 3D

Como apresentado anteriormente na metodologia, as paredes da planta baixa foram criadas utilizando a *API OpenGL* em *Python*. Antes de criar o modelo 3D, é feito uma

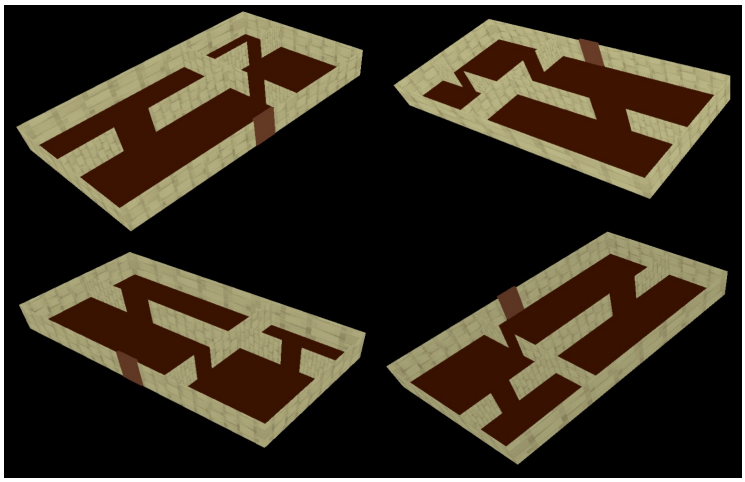
identificação de aberturas semelhante a identificação de aberturas descrita na seção 4.2, só que nesse caso à identificação de aberturas será feita somente nas paredes externas da planta, para então posicionar as portas. Podemos observar os resultados da geração do modelo 3D com paredes e portas das plantas testadas nas figuras 26 e 30.

Figura 26 – Resultado da geração 3D das paredes e portas do modelo da primeira planta baixa de testes.



Fonte: o autor

Figura 27 – Resultado da geração 3D das paredes e portas do modelo da planta baixa obtida na 40ª Feira do Livro de Alegrete 2019.



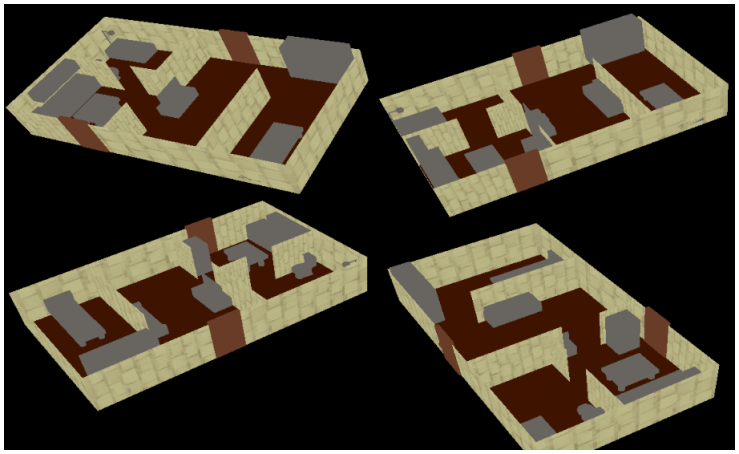
Fonte: o autor

5.2.2 Posicionamento dos móveis na planta 3D

Como dito na seção 4.3, os móveis serão posicionados com base nos pontos dos cantos de cada cômodo, subtraindo ou somando valores conforme as dimensões do móveis,

para evitar que não haja sobreposição dos móveis com as paredes, como foi mostrado na Figura 15. Podemos observar o resultado do posicionamento correto dos móveis da primeira planta de testes na Figura 28. Nesse primeiro a maioria dos móveis foi posicionado

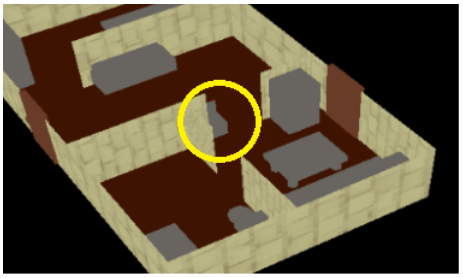
Figura 28 – Resultado da geração 3D com os móveis posicionados nos cômodos na primeira planta baixa usada nos testes.



Fonte: o autor

de forma correta, sem nenhuma sobreposição com paredes ou outro móveis, no entanto houve um pequeno erro com um dos móveis da sala, onde uma parte do móvel ficou bloqueando uma abertura, como podemos ver na Figura 29. Este erro ocorre pelo fato de que o algoritmo possui somente uma configuração de móveis para cada cômodo, já que no estado atual da ferramenta o algoritmo não leva em consideração as aberturas do modelo, os móveis vão sempre ser posicionados na mesma forma, podendo haver o bloqueio parcial ou completo de alguma abertura. Neste primeiro exemplo, o móvel que está bloqueando a abertura sempre será posicionado junto à parede, exatamente no meio do cômodo, onde coincidentemente existe uma abertura.

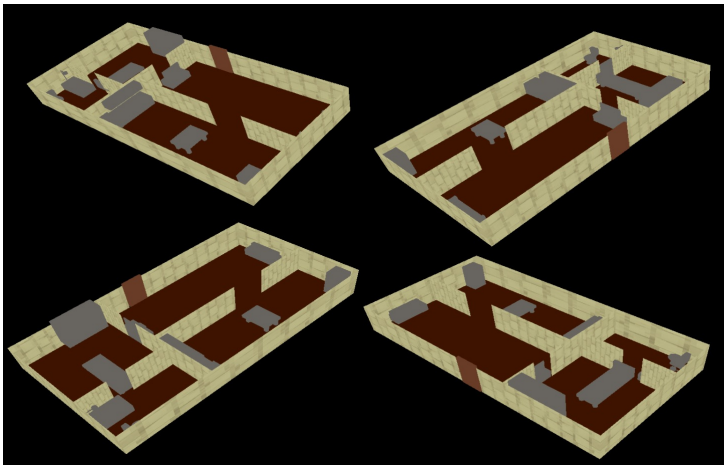
Figura 29 – Móvel bloqueando abertura.



Fonte: o autor

O segundo teste foi o posicionamento dos móveis na planta baixa desenhada a mão por crianças de 12 a 16 anos durante a 40ª Feira do Livro de Alegrete, o resultado pode ser observado na Figura 30.

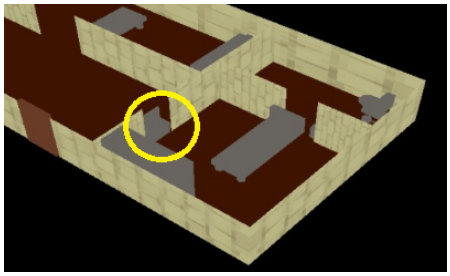
Figura 30 – Resultado da geração 3D com os móveis posicionados nos cômodos na planta baixa obtida na 40ª Feira do Livro de Alegrete.



Fonte: o autor

Os resultados desse segundo modelo foram praticamente os mesmos do primeiro teste. Os móveis foram posicionados corretamente em seus respectivos cômodos e sem sobreposição dos móveis com as paredes. Porém, um dos móveis também bloqueou uma das aberturas do cômodo, como podemos observar na Figura 31.

Figura 31 – Móvel bloqueando abertura



Fonte: o autor

5.2.3 Análise dos resultados

Nesta seção serão apresentados os resultados do funcionamento geral da ferramenta. Os resultados foram classificados em três tipos, são eles: satisfatórios, médios e ruins, onde os resultados satisfatórios são aqueles em que o algoritmo conseguiu identificar

corretamente todas as paredes da planta baixa, gerar o modelo 3D e também posicionar os móveis de forma correta; resultados médios são aqueles em que o algoritmo consegue identificar e criar de forma correta todas as paredes da planta baixa, porém, com erros no posicionamento do móveis, como por exemplo, sobreposição de um móvel com outros móveis ou paredes, ou então um bloqueio completo de uma das aberturas; resultados ruins são aqueles em que o algoritmo não conseguiu identificar ou corrigir todas a paredes da planta baixa. Os resultados foram satisfatórios em 58% dos testes, resultados médios em 25% e resultados ruins em 17% dos testes. As retas foram identificadas e corrigidas de forma satisfatórias em casos onde as plantas baixas são de fato plantas baixas mais parecidas com a realidade. Porém, em alguns casos como na identificação de retas em uma imagem com muitas retas próximas entre si como por exemplo em um labirinto, ainda podem ocorrer erros de detecção de retas, cômodos ou no posicionamento dos móveis, pois os algoritmos desenvolvidos neste trabalho utilizam proporções inerentes à cômodos em imóveis comuns. Alguns problemas como a baixa qualidade da imagem digitalizada em termos de desenho ou parâmetros de digitalização como iluminação, reflexo ou ângulo, por exemplo, são uma limitação adicional por estarem ligados diretamente à atuação do usuário. Os resultados em relação a geração do modelo 3D da planta baixa, são os mesmos resultados da detecção e correção de retas, já que a geração das paredes 3D depende diretamente das retas obtidas nas etapas anteriores. Logo, quando a detecção e correção de retas apresentar erros, a geração do modelo 3D irá apresentar os mesmo erros. O posicionamento dos móveis também teve um impacto para a porcentagem de resultados médios, pelo fato de móveis posicionados de maneira incorreta dentro dos cômodos, onde o móvel acaba bloqueando parcialmente ou completamente alguma abertura do cômodo, como por exemplo na Figura 31.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresenta o desenvolvimento de uma nova ferramenta, capaz de criar modelos de construções civis em uma perspectiva 3D, e também inserir objetos em seus interiores a partir de uma imagem de uma planta baixa feita a mão livre, diferentemente de outras ferramentas que necessitam de uma planta baixa impressa em um determinado formato ou desenhada com instrumentos de precisão. Dessa forma, acreditamos que a ferramenta proposta neste trabalho traz novas e interessantes características, proporcionando grande facilidade e simplicidade ao usuário que deseja visualizar rapidamente uma primeira versão de um imóvel, com resultados praticamente idênticos aos de ferramentas que exigem uma entrada padronizada. Para o reconhecimento de retas foi utilizado a técnica de Hough da biblioteca *OpenCV* que, apesar de limitada, se mostrou adequada com as devidas adaptações envolvendo as etapas de filtragem desenvolvidas. As retas detectadas foram corrigidas por métodos criados especificamente para cada um dos problemas encontrados na detecção da transformada de Hough. Para a geração das paredes e móveis do modelo 3D da planta baixa, foi utilizado o *OpenGL* da biblioteca *PyOpenGL*.

6.1 Trabalhos futuros

Para trabalhos futuros busca-se tornar a ferramenta mais robusta para suportar uma quantidade mais diversa de plantas baixas, mesmo que sejam improváveis. O objetivo dessa melhoria é assegurar que, para os casos mais comuns, a detecção de retas consiga detectar, com maior garantia e de forma correta, todas as retas da planta. Além disso, busca-se implementar uma IA (inteligência artificial) para a geração de um posicionamento mais inteligente dos móveis dentro dos cômodos, com a possibilidade de gerar diferentes configurações de móveis e também diferentes números de cômodos do mesmo tipo. Busca-se, também, uma melhora na qualidade da visualização do resultado final, aplicando uma mapeamento de texturas não só nas paredes como é o estado atual da ferramenta, mas também nos demais elementos da planta, como no chão, portas e móveis do modelo 3D.

REFERÊNCIAS

AHMED et al. Improved automatic analysis of architectural floor plans. In: IEEE. [S.l.], 2011. Citado na página 21.

AHMED, S. et al. Automatic analysis and sketch-based retrieval of architectural floor plans. Elsevier, v. 35, p. 91–100, 2014. Citado na página 20.

CABELLO, R. **Three.js**. 2010. Disponível: https://threejs.org/editor/. [Online; acessado em 27-05-2019]. Citado na página 40.

CAMOZZATO, D. et al. Procedural floor plan generation from building sketches. Springer, v. 31, p. 753–763, 2015. Citado 2 vezes nas páginas 19 e 23.

CANNY, J. A computational approach to edge detection. In: **Readings in computer vision**. [S.l.]: IEEE, 1986. Citado na página 26.

HOUGH, P. Method and means for recognizing complex patterns. In: . [S.l.: s.n.], 1962. Citado na página 15.

INC., A. **AUTOCAD**. 1982. Disponível: https://www.autodesk.com.br/. [Online; acessado em 01-06-2019]. Citado na página 43.

KAUFMAN, A. et al. Volume graphics. IEEE, v. 26, p. 51–64, 1993. Citado na página 16.

LEARNED-MILLER. Introduction to computer vision. In: . [S.l.: s.n.], 2011. Citado na página 13.

LLADOS, J.; LOPEZ-KRAHE, J.; MARTI, E. A system to understand hand-drawn floor plans using subgraph isomorphism and hough transform. Springer, v. 10, p. 150–158, 1997. Citado na página 21.

MANSSOUR, I. H.; COHEN, M. Introdução à computação gráfica. researchgate, v. 13, p. 43–68, 2006. Citado 2 vezes nas páginas 16 e 41.

MERRELL, P.; SCHKUFZA, E.; KOLTUN, V. Computer-generated residential building layouts. In: ACM. [S.l.], 2010. Citado na página 22.

MESSMER, B. T.; BUNKE, H. Automatic learning and recognition of graphical symbols in engineering drawings. In: SPRINGER. [S.l.], 1995. Citado na página 21.

OKORN, B. et al. Toward automated modeling of floor plans. In: VANDERBILT UNIVERSITY. [S.l.], 2010. Citado na página 20.

OPENCV. **OpenCV Wiki**. 2000. Disponível: https://opencv.org/. [Online; acessado 08-06-2018]. Citado na página 26.

OPENGL. **Wiki do OpenGL**. 2018. Disponível: https://www.khronos.org/opengl/wiki/. [Online; acessado em 23-05-2018]. Citado na página 41.

SANTOS, C. **Planta Baixa Simples de Casas**. 2015. Disponível: https://www.tudoconstrucao.com/plantas-de-casas-modelos-planta-baixa-projetos/. [Online; acessado em 10-04-2018]. Citado na página 13.

SOBEL, I. An isotropic 3 x 3 gradient operator. Stanford Artificial Project, p. 271–272, 1968. Citado na página 20.

TOMASI, C.; MANDUCHI, R. Bilateral filtering for gray and color images. IEEE International Conference on Computer Vision, v. 98, p. 2, 1998. Citado na página 19.

WRIGHT, R. S.; SWEET, M. **OpenGL SuperBible with Cdrom**. [S.l.]: Sams, 1999. Citado na página 41.