PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E PROCESSOS INDUSTRIAIS – MESTRADO ÁREA DE CONCENTRAÇÃO EM CONTROLE E OTIMIZAÇÃO DE PROCESSOS INDUSTRIAIS

Tiago Zonta

DESENVOLVIMENTO DE ALGORITMOS COMPUTACIONAIS PARA CRIAÇÃO E ORÇAMENTO DE PROJETOS DE FORROS

Santa Cruz do Sul, Dezembro de 2009

Tiago Zonta

DESENVOLVIMENTO DE ALGORITMOS COMPUTACIONAIS PARA CRIAÇÃO E ORÇAMENTO DE PROJETOS DE FORROS

Dissertação apresentada ao Programa de Pós-Graduação em Sistemas e Processos Industriais – Mestrado, Área de Concentração em Controle e Otimização de Processos Industriais, Universidade de Santa Cruz do Sul, como requisito parcial para obtenção do título de Mestre em Sistemas e Processos Industriais.

Orientador: Prof. Dr. João Carlos Furtado

Co-orientador: Prof. Dr. Itamar Leite de Oliveira

Santa Cruz do Sul, Dezembro de 2009

AGRADECIMENTOS

Agradeço a Deus pela vida e pela saúde, e que proporciona que eu consiga passar por estes momentos.
Agradeço aos meus avós Plínio e Maria, Geraldo e Rosa (*in memorian*). Tenho certeza que sempre estão de olho me protegendo e me mostrando o melhor caminho a ser seguido.
Agradeço aos meus pais Jesué e Inês e aos meus irmãos Pierre e Josué, que sempre me apoiaram em qualquer uma das minhas caminhadas. Muitas vezes, ver o orgulho deles vale mais do que qualquer coisa, amo muito todos eles.
Agradeço especialmente minha esposa Juliana que amo tanto por, novamente, me ajudar, compreender e mostrar o quanto ela é importante para tudo o que represento.
Não poderia esquecer os mestres. Não somente os professores, mas também os colegas e pessoal da secretaria que, às vezes, mesmo não estando em sala, estão nos ensinando algo, sempre prontos a ajudar. Aos professores, minha admiração por colaborarem com características diferentes: professor Panta, maestria em matemática e atenção aos alunos; Rolf, postura a ser seguida em sala; Liane e Adilson, dicas muito importantes; Rejane, inteligência a toda prova; Jacques, motivação; e professor Ferrão, mesmo não participando de suas aulas suas dicas em bancas e seminários foram de grande valia.
Agradeco, especialmente, ao meu orientador João, por sempre demonstrar calma e confiar que o trabalho estava sendo realizado da melhor forma possível, e também por publicar um trabalho que cito desde minha graduação.
Um agradecimento especial, também, ao professor Itamar, que novamente me ajudou com sua sabedoria, conhecimento e eterna amizade. Muito da minha postura como professor hoje vem do que aprendi com ele, que sempre valorizou o aluno e nunca se negou a dar instruções.

RESUMO

As empresas possuem vários processos para redução de custos, que são freqüentemente obtidos a partir de estratégias aplicadas na produção, armazenagem, montagem e distribuição. Neste sentido um problema que pode ser apontado é a perda de material, que ocorre devido à dificuldade de encontrar a melhor forma de fazer um reaproveitamento.
Muitos materiais são distribuídos pelas indústrias de matéria prima em tamanho padrão e devem ser cortados em novos itens de tamanho igual ou menor que o original. No momento do corte, muitos objetos menores não são utilizados. Essa inutilização, quando em grande quantidade, acarreta um desperdício de material muito grande e, consequentemente, um problema financeiro.
Estas empresas, se corretamente instruídas, poderiam investir em pesquisas que pudessem oferecer soluções rápidas e com melhor resultado, utilizando a alta capacidade de processamento dos computadores, já que muitas delas possuem soluções manuais.
O problema, então, é achar a melhor forma de fazer este corte de modo a minimizar a perda do objeto a ser cortado ou maximizar o aproveitamento de material dos itens utilizados. Dessa forma, o foco desta dissertação foi estudar e desenvolver um método para resolução de um problema encontrado numa empresa de montagem de ambientes de paredes de divisórias e forros.
A montagem dinâmica de ambientes é uma solução cada vez mais utilizada por quem necessita de mudanças constantes nos *layouts* de seus ambientes, evitando obras que tragam transtornos. Esses podem ser denominados como ambientes facilmente modificáveis. Toda esta montagem é feita a partir de desenhos pré-definidos, nos quais são determinadas as dimensões da obra, permitindo o calculo do orçamento. Então, porque não utilizar também as informações do desenho para entrada do problema do corte?
Este trabalho apresenta a criação destes desenhos através de um componente CAD, a utilização de grafos planares e geometria computacional com a biblioteca CGAL na sua interpretação, gerando, assim, as informações necessárias para a pesquisa operacional aplicar sua característica de resolver problemas realizando o preenchimento ou a distribuição dos materiais obtendo os itens para aplicação do corte unidimensional.
A contribuição deste trabalho é mostrar todo este método que, até então, era aplicado de forma específica ou de forma separada, isto é, uma solução responsável pelo desenho e outra que recebe a entrada manualmente das dimensões para criação do projeto.

Palavras-chave: grafos planares, geometria computacional, biblioteca CGAL, pesquisa operacional.

ABSTRACT

The companies have several procedures to reduce production and storage costs, which are frequently obtained through strategies applied at production, assembly and distribution. In this way, a problem that can be pointed out is the material loss, which happens due to the difficulty faced by these companies in order to look for the best way to reuse.
A huge amount of materials are distributed by the raw material industries in a standard size which, in order to be used, must be cut in new smaller or equal size items. When this cutting happens, several smaller objects are not used. This destruction, when in large quantities, leads to a huge material loss and, consequently, to a financial break.
These companies, if properly directed, could invest in researches that could provide faster and better-resulted solutions by using the computers' high processing capability, once many of them still have manual solutions.
The problem, then, is to find the best way to make this cutting in order to minimize the object to be cut's loss or to maximize the material use from the resorted items. Thus, this work's main goal was to study and to develop a method to solve a problem faced by a company whose market is to assemble environments with wall partitions and ceilings.
The dynamic environments' assembly is a solution increasingly used for those who need to change their environment's layout constantly, avoiding, then, constructions that may bring inconveniences. These ones can be termed as easily changeable environments. The whole assembly is done by pre-set drawings in which all the construction's dimensions are determined, allowing, then, the construction's budget calculation. So, why wouldn't it also be possible to use the drawing's information or the cutting problem's entry?
This work presents these drawings' creation through a CAD component, planar graphs and computational geometry usage with CGAL library in their interpretation, generating, thus, the whole necessary information the operational research needs to apply its features in solving problems by fulfilling or distributing the material and obtaining the items to apply the one-dimensional cutting.
This work's contribution is to present this whole method that, theretofore, was applied in a specific or separated form, it means, one solution that is responsible for the drawing and another that receives the dimensions' entry manually in order to create a project.

Keywords: planar graphs, computational geometry, CGAL Library, operational research.

LISTA DE ILUSTRAÇÕES

LISTA DE MODELOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACM – *Association for Computing Machinery*.

CA – Complexidade de Algoritmos.

CAD – *Computer Aided Design* (Projeto auxiliado por computador).

CGAL – *Computational Geometry Algorithms Library* **.**

FFD - Algoritmo *First Fit Decreasing*.

GC – Geometria Computacional.

AG – Algoritmo Guloso

GUI – *Graphical User Inteface*.

LD – *Limitante de Dantzig.*

PC – Problema do Corte.

PCE – Problema do Corte e Empacotamento.

PCU – Problema do Corte Unidimensional.

PO – Pesquisa Operacional.

PVC – Poli Cloreto de Vinila

SIGGRAPH - *Special Interest Groups Computer Graphics and Interactive Techniques.*

STL – *Standard Template Library*.

SUMÁRIO

1 INTRODUÇÃO

A pesquisa operacional (PO) é conhecida por apresentar métodos para resolução e explicação de problemas com grande complexidade, obedecendo às restrições por estes impostas. Neste trabalho será apresentado um método para resolução de um problema que, atualmente, é realizado de forma manual e acaba gerando perda de tempo e de material.

Segundo Arenales et al (2007), a observação e descrição de fenômenos é uma forma de se fazer ciência. Sempre, a partir da observação, consegue-se identificar regras e leis que podem ser descritas por relações matemáticas.

A Figura 1 mostra um diagrama que ilustra uma abordagem simples de solução de um problema que utiliza modelagem matemática e define, basicamente, os passos necessários para identificação das ações necessárias para realização deste trabalho.

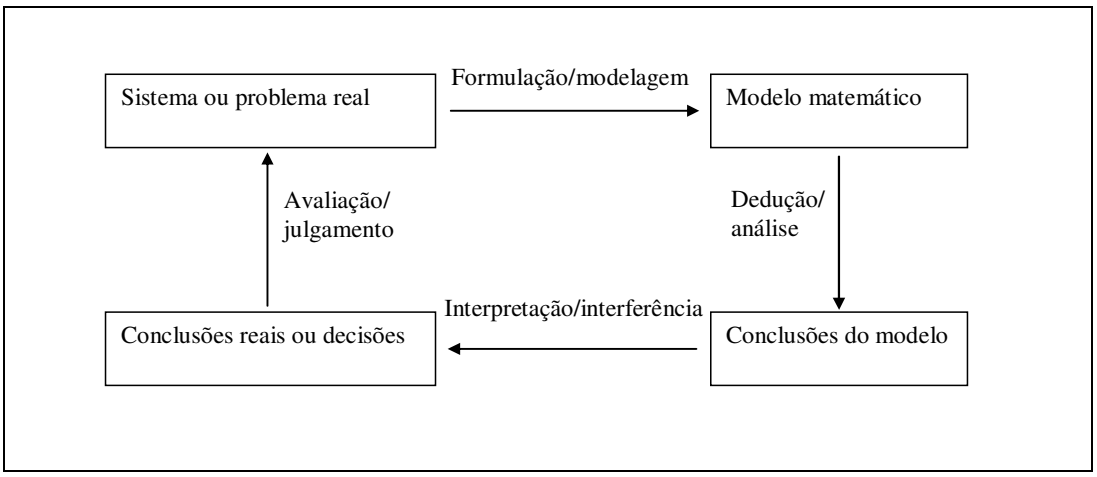

Figura 1. Processo de modelagem.
Fonte: ARENALES et al, 2007.

O sistema ou problema real apresentado aqui segue a idéia de que um objeto deva ser cortado, ao longo de seu comprimento, em itens (pedaços) de comprimentos especificados. Cada item tem um valor associado chamado de valor de utilidade. Itens cujos comprimentos

não foram especificados são considerados perdas e têm valores de utilidade nulos. Surge, então, um problema de otimização combinatória: Maximizar valor de utilidade total.

O problema, então, é achar a melhor forma de cortar o objeto a fim de produzir tais itens de modo que o valor de utilidade total seja máximo (equivale a dizer que a perda de material seja mínima). O problema do corte e empacotamento, como é chamado, é considerado de alta complexidade computacional, teoricamente classificado como Não Polinomial (NP) e encontrado em várias dimensões (ARENALES et al, 2007; YANASSE, 1997). No trabalho em questão será abordado o corte de uma dimensão e será proposto um estudo de caso para sua aplicação, tornando fácil a visualização de que sua utilização, na prática, pode trazer diversas vantagens econômicas.

O estudo de caso baseia-se na realização do desenho de uma planta baixa de uma obra, a fim de determinar quantos metros quadrados ela terá e, com isso, obter a quantidade de materiais necessários, realizando o preenchimento da área desenhada com os materiais desejados. Após a realização deste preenchimento, podem-se obter os itens que abastecerão as entradas necessárias para a utilização do problema descrito acima. A planta baixa representará uma obra de montagem de forros de PVC e modulares.

1.1 Objetivo Geral

O objetivo geral deste trabalho é propor um método para resolução do problema de montagem de forros de PVC e modulares englobando o desenho e a determinação de seu preenchimento com os materiais escolhidos, a fim de determinar, da melhor forma, o corte dos materiais e fornecer, prontamente, o orçamento automático.

1.1.1 Específicos

Como objetivos específicos, citam-se:

Realizar o levantamento dos requisitos e materiais necessários para a montagem do forro em conjunto com a empresa responsável pelo estudo de caso;

Realizar estudo do problema e suas variações a partir da pesquisa bibliográfica e experimental, descrevendo, assim, as abordagens existentes;

Estudar o componente (TCAD XP, 2008) a ser adquirido e realizar as diversas formas geométricas que existem em desenhos de planta baixa;

Realizar a identificação e separação das figuras geométricas tratando a planta baixa como um todo ou dividida em cômodos;

Realizar o processo de preenchimento da planta baixa desenhada, obedecendo as dimensões e padrões de montagem. Utilizar os itens gerados no preenchimento para aplicação do problema do corte, maximizando o aproveitamento dos materiais e minimizando o tempo de montagem da obra.

1.2 Justificativa

No ponto de vista comercial, os problemas relacionados com reaproveitamento são essenciais para o planejamento da produção em algumas indústrias, tais como indústrias de papel, vidros, metalúrgica, plástica, têxtil, logística etc. Nestas indústrias, a redução dos custos de produção, armazenagem etc, é freqüentemente obtida utilizando estratégias na montagem, distribuição ou até agregando ao custo do produto. Estas empresas, se corretamente instruídas, poderiam investir em pesquisas que pudessem oferecer soluções rápidas e com melhor resultado utilizando os computadores.

Já no ponto de vista científico, estes problemas são considerados como de alta complexidade computacional por serem altamente combinatórios, cuja solução ótima, em muitos casos, ainda está limitada a pequenas instâncias. Consequentemente, a solução destes problemas tem sido um desafio constante para os pesquisadores de diversas áreas, principalmente, Ciência da Computação, Matemática e Engenharias.

A justificativa da possibilidade de criação do método proposto por este trabalho é a ligação destes dois pontos de vista. Do ponto de vista comercial foi observado pelo proprietário da empresa Eocaplan (2008), que o processo atual utiliza o desenho da obra para oferecer as informações necessárias para criação do projeto de montagem de um ambiente facilmente modificável. Do ponto de vista científico, surgiu a análise de como isso pode ser realizado diminuindo o processo manual existente. O desenho oferece as informações necessárias no processo manual e a ciência oferece metodologias para o reaproveitamento de material, então será proposto um método que automatize a criação de um projeto de ambientes facilmente modificáveis.

A visualização da possibilidade da aplicação deste trabalho foi que, atualmente, alguns segmentos da indústria, como confecção e têxtil, utilizam sistemas que aplicam o reaproveitamento de material, por exemplo, sistemas desenvolvidos pelas empresas Audaces (2009, http://www.audaces.com/novo/pt/home/) e RZ Sistemas (2009, http://www.rzsistemas.com.br/int/rzcadtextil.php?txt=1). Para ambientes facilmente modificáveis, algumas tentativas foram feitas pelas empresas Eucatex (2009, http://www.eucatex.com.br/) e Sul Módulos (2009, http://www.sulmodulos.com.br/index.htm) utilizando a metragem da obra para calculo aproximado do orçamento.

Outro fator que justifica o objetivo deste trabalho é que, a parte referente às paredes de divisórias já foi realizada pelo autor na graduação e está em funcionamento até o momento. O trabalho aqui em questão apresenta uma complexidade maior que o das paredes de divisórias. Esta complexidade está relacionada com o aumento de formas geométricas que uma planta baixa oferece, muito diferente das paredes, que apresentam somente retângulos.

1.3 Método

Este trabalho originou-se a partir de uma pesquisa exploratória, pesquisa descritiva e uma pesquisa experimental.

Para atingir os objetivos específicos, este trabalho foi dividido em fases. O estudo teve início com o levantamento dos requisitos junto à empresa na qual o estudo de caso foi realizado. Nesta interação, foram verificados os materiais necessários para realização da montagem dos diversos tipos de forros. Esse levantamento demonstrou um padrão de montagem, auxiliando na decisão de como modelar a forma de configuração da base de dados de cada tipo ou marca de material. Isso faz com que seja possível criar uma ferramenta computacional que não seja exclusiva da empresa estudada. Além disso, a ferramenta oferecerá a possibilidade de apresentar um mesmo projeto com orçamentos diferenciados para cada tipo de material ou marca.

Com os requisitos e vários exemplos de obras em mãos, foi iniciado o estudo para criação da planta baixa utilizando o componente TCAD da ferramenta de desenvolvimento Delphi. O levantamento mostrou que o problema maior não seria apenas criar o desenho, e sim, dividí-lo em diversos polígonos, identificando nele a dimensão total e os cômodos individualmente. Com isso, o trabalho apresentou outro desafio, o estudo de grafos e geometria computacional para a criação de algoritmos e estruturas de dados que sejam capazes de separar (mapeamento), armazenar e recuperar as partes constituintes da planta baixa referente a um projeto de forros. A complexidade das estruturas de dados necessárias e da implementação de algoritmos geométricos, devido à precisão e robustez numérica, fez com que fosse usada uma biblioteca que continha tais estruturas de dados e algoritmos já implementados e amplamente testados, a CGAL (*Computational Geometry Algorithms Library*). Tal biblioteca é *Open Source* e possui uma grande quantidade de algoritmos e estruturas de dados, inclusive aqueles necessários para este trabalho. Outro aspecto importante da CGAL é que trata os casos degenerados que ocorrem com frequência na geometria computacional (O'ROURKE, 1994).

Com a realização dos desenhos e a sua separação em ambientes será possível construir o processo de preenchimento de forma dinâmica de acordo com a escolha do material, obtendo, assim, os itens necessários para a aplicação do corte. Como a aplicação do corte já foi previamente estudada em Zonta (2000.1; 2005), uma técnica híbrida já foi criada, na qual, primeiramente, o corte é resolvido com uma dimensão até finalizar os itens que se encaixam em suas características.

O método descrito assemelha-se muito com os requisitos necessários para utilização da pesquisa operacional segundo Gabriel R. Britan citado em Arenales et al (2007, p. XI)

> Três requisitos são necessários para utilização da pesquisa operacional. O primeiro envolve a compreensão de características e atributos de um sistema complexo e a habilidade de abstrair e traduzir os pontos mais importantes em um modelo matemático ou de simulação. O segundo consiste da habilidade para desenvolver métodos de resolução para modelos e utilizar pacotes comerciais com conhecimento sobre os métodos utilizados neste. O terceiro envolve a comunicação com clientes para compreender o problema e explicar resultados não intuitivos, mas importantes, gerados pela aplicação de pesquisa operacional.

1.4 Estrutura do Trabalho

Este trabalho está organizado em 6 capítulos, os quais abordam conceitos e estratégias para apresentação e criação de projetos de forros:

Capítulo 2 – Apresenta os ambientes. Tais ambientes serão chamados aqui de facilmente modificáveis por causa de suas características. Mostra o método hoje utilizado para realização de um projeto, que envolve desenho, cálculo e definição do orçamento. O capítulo exibe exemplos de cada tipo de forro;

Capítulo 3 – Apresenta as teorias estudadas para realização dos desenhos, mapeamento e armazenamento das figuras geométricas resultantes da planta baixa. Neste capítulo, foram mantidos os estudos de grafos planares e geometria, mesmo com a utilização da biblioteca CGAL, que também é apresentada;

Capítulo 4 – Apresenta a teoria responsável por despertar a idéia deste trabalho. Neste capítulo é apresentada a idéia da aplicação do corte unidimensional a partir dos dados resultantes dos desenhos geométricos;

Capítulo 5 – Resultados e comparações do modelo criado. Neste capítulo é apresentado e discutido a metodologia que utiliza geometria computacional e otimização para projeto e orçamento de forrros;

Capítulo 6 – Conclusão e trabalhos futuros.

2 AMBIENTES FACILMENTE MODIFICÁVEIS

Qual seria a vantagem de construir ambientes que possibilitam a montagem e desmontagem conforme a necessidade?

Esta pergunta, fora do Brasil, é respondida com facilidade e, nos Estados Unidos, é altamente importante. Lá, são encontradas muitas construções, principalmente casas, feitas de forma rápida e barata, utilizando produtos derivados da madeira. No Brasil, este conceito para casas ainda não é muito utilizado, mas para ambientes comerciais e industriais já é uma realidade. Imagine um escritório que, toda vez que uma modificação de *Layout* fosse necessária, paredes tivessem que ser destruídas e, consequentemente, o forro e outros acabamentos envolvidos refeitos.

O trabalho aqui proposto tem relação com outro já realizado durante graduação do autor (ZONTA 2000), na qual foi desenvolvido um sistema de desenhos, corte e orçamento de paredes de divisórias, ver Figura 2. As divisórias, atualmente, são facilmente encontradas em hospitais, escolas, sanitários, shoppings, consultórios, farmácias, frigoríficos, academias de ginásticas, indústrias alimentícias, laboratórios, escolas e outros. Estas aplicações podem variar de acordo com as necessidades.

Figura 2. Exemplo de Paredes de divisórias.
Fonte: EOCAPLAN, 2008.

Num projeto de ambientes facilmente modificáveis, além de paredes divisórias, há também a necessidade de montagem do forro. Existem hoje dois tipos de forros: modulares e não modulares, ambos de PVC. Obviamente, a empresa na qual será feito o estudo de caso vem trabalhando com ambos.

Todo o processo de montagem de um forro se inicia com o projeto (desenho) e deste determina-se a área, em metros quadrados, que a obra terá. Com estas informações é possível realizar do orçamento final da obra.

A seguir, serão apresentados com mais detalhes os componentes de um projeto de forro e, com isso, um melhor entendimento do objeto de estudo deste trabalho.

2.1 Forros Modulares

O forro modular utiliza placas removíveis facilitando o acesso acima do forro e ainda possibilita o uso de luminárias embutidas ou externas, ver Figura 3.

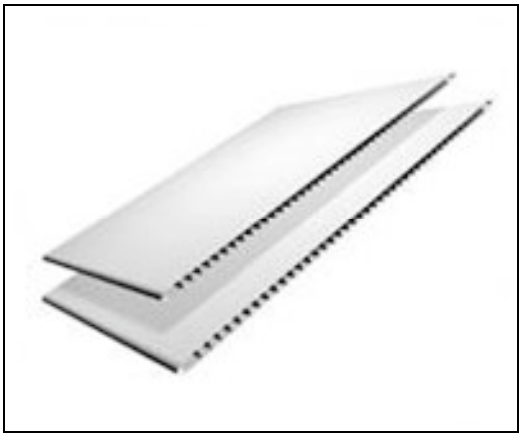

Figura 3. Forro de PVC modular.
Fonte: EOCAPLAN, 2008.

O forro modular é montado sobre uma estrutura formada de perfis, conforme mostra a Figura 4. O material, placa de PVC, é devidamente encaixado sobre esta estrutura que é montada de acordo com o tipo de forro.

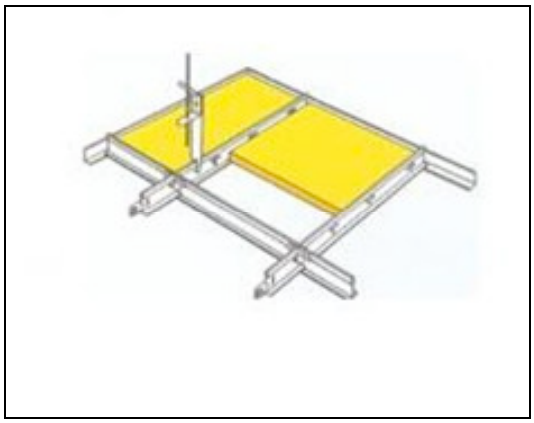

Figura 4. Estrutura para montagem dos forros.
Fonte: EOCAPLAN, 2008.

A estrutura da Figura 4 é conhecida como sistema “T” leve, com perfis em aço galvanizado e pintura em epóxi-poliéster pó ou com perfis em alumínio anodizado natural.

Um projeto de montagem é realizado da seguinte forma:

É necessário saber a medida do rebaixe até o nível do forro para saber a metragem dos pendurais;

É feita a distribuição das placas, utilizando-se o *AutoCAD*, em toda a área a ser forrada para, assim, ser apurada a quantidade das mesmas;

Com a modulação desenhada é possível levantar, também, a quantidade dos perfis para a grade e para as cantoneiras;

As presilhas são três vezes a quantidade de placas;

O valor da mão-de-obra é baseado na metragem quadrada.

2.2 Forros não Modulares

O forro não modular possui revestimento feito a base de cloreto de polivinila. Composto de painéis lineares de fácil instalação, ver Figura 5.

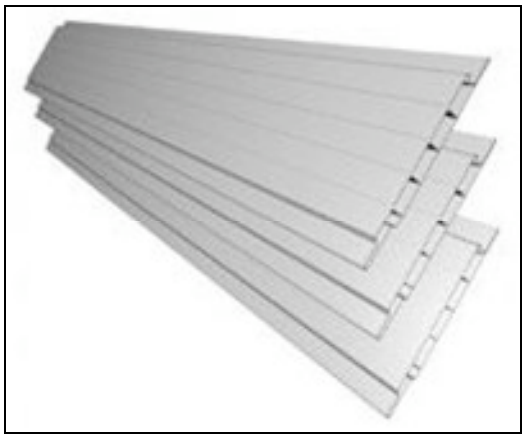

Figura 5. Forro de PVC não modular.
Fonte: EOCAPLAN, 2008.

A estrutura necessária para sua montagem é descrita a seguir:

Grade necessária, composta por perfis rígidos RG-26/0.50 em aço galvanizado tipo "cartola";

Perfil Pontalete: fixado com parafusos de aço e rebites na posição vertical em estrutura metálica ou laje;

Perfil Mestre: fixado com rebites galvanizados no Perfil Pontalete na posição horizontal a cada 1,50m;

Perfil Travessa: fixado com rebites galvanizados a cada 0,60m no sentido inverso do Perfil Mestre.

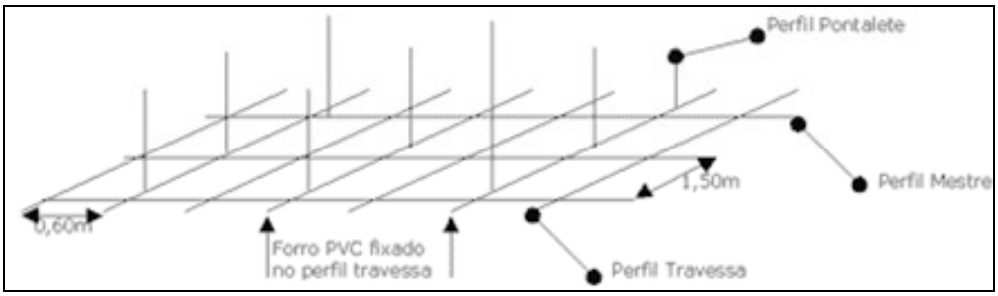

Figura 6. Estrutura para forros não modulares.
Fonte: EOCAPLAN, 2008.

Um projeto de montagem é realizado da seguinte forma:

É necessário saber a medida do rebaixe até o nível do forro para saber a metragem dos pendurais;

Os arremates são a soma do perímetro individual de cada ambiente;

Dependendo da largura da régua de forro PVC utilizada (100 ou 200mm) e baseado na largura da área a ser forrada será encontrada a quantidade de placas e o comprimento de melhor aproveitamento;

A quantidade de rebites é de uma proporcional a metragem da obra. Neste caso para cada 100 m² utiliza-se 2.000 unidades;

O valor da mão-de-obra é baseado na metragem quadrada.

2.3 Considerações

Com os dados levantados junto à empresa foi possível visualizar a possibilidade da realização do projeto partindo de um desenho. No processo atual, o desenho oferece diversas informações para a criação do projeto do forro e independente do tipo, sempre é necessária a construção da planta baixa da obra toda. Nos próximos capítulos, serão mostrados os recursos e conceitos computacionais para o desenvolvimento e construção informatizados de um projeto de forro.

3 DESENHOS GEOMÉTRICOS

Esse capítulo tem por objetivo apresentar o estudo de técnicas computacionais que podem ser utilizadas para obter informações a partir de desenhos ou projeto do forro, isto é, através da planta baixa, ver Figura 7. Estas fornecerão as informações para o problema do corte, que só será executado após determinado os itens necessários para montagem do forro. Este processo, na maioria dos sistemas automáticos, é feito a partir de informações relativas às dimensões dos ambientes ou até às dimensões de cada item a ser cortado, tornando-se mais desgastante do que realizar o corte empiricamente, obedecendo aos padrões dos materiais utilizados. O grande diferencial que será proposto, parte da idéia de obter todas as informações necessárias através da iteração com o usuário, o qual terá apenas a tarefa de realizar o desenho que deseja orçar.

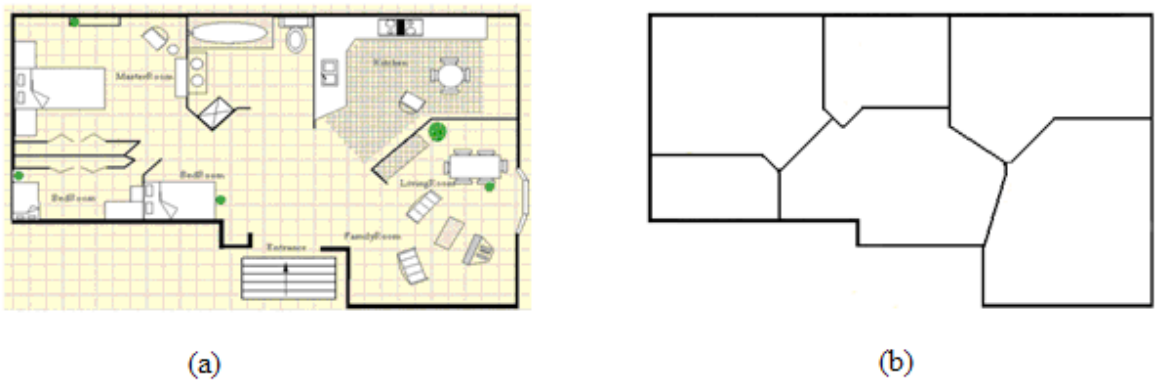

Figura 7. (a) Planta baixa de uma obra trazida pelo cliente. (b) Planta baixa desenhada a partir de (a) para orçamento e projeto de forro.
Fonte: http://www.codeidea.com/, 2008.

Conforme a Figura 7(b), a planta baixa mostra o desenho como um todo, é necessário saber a geometria e quantos cômodos (ambientes), além de suas dimensões, formam a planta. Para a identificação de cada ambiente será necessário o desenvolvimento de algoritmos e estruturas de dados específicos para este tipo de aplicação, já que, para o computador, um desenho é basicamente uma estrutura de dados. Para isso, foram estudadas duas técnicas para auxiliar na solução deste problema: a) grafos planares; e b) geometria computacional. Estas duas técnicas serão abordadas nas próximas seções.

A Figura 8 apresenta o exemplo de uma planta baixa composta por 5 cômodos. Neste exemplo, será necessário identificar todos os 5 polígonos; considerando evidentemente que a entrada foram os 3 retângulos mostrados. A identificação de cada polígono possibilitará que eles sejam trabalhados de forma independente considerando a forma, o tipo e o forro que será aplicado em cada cômodo. A Figura 9 mostra dois objetivos deste trabalho; 1) a realização do desenho utilizando o componente TCAD; 2) identificar os polígonos, possibilitando a configuração separada de cada cômodo da planta baixa.

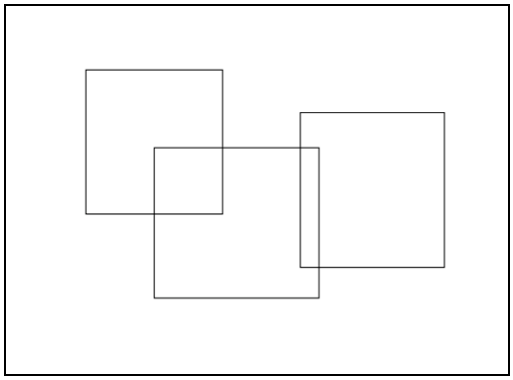

Figura 8. Planta baixa criada com três polígonos.

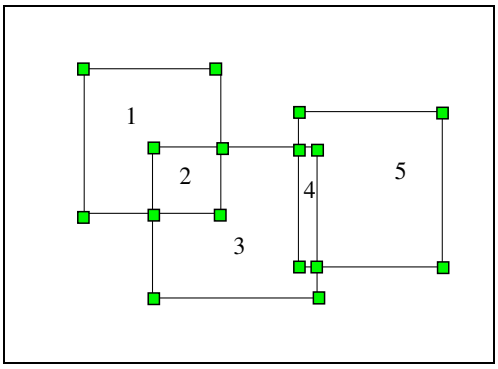

Figura 9. Planta baixa identificando, separadamente, cada um dos polígonos (1 a 5).

3.1 Grafos

Grafos são conhecidos como estruturas compostas por vértices ligados por arestas. Na geometria, poderia-se fazer analogia com os pontos e segmentos de reta. Um grafo pode ser representado matematicamente por ***G = (V,E)***, na qual o conjunto de vértices é representado por ***V*** e suas ligações, chamadas de arestas, por ***E***. No grafo da Figura 10 podem ser vistos 6 vértices (***V = {1,2,3,4,5,6}***) representados pelos círculos numerados e com arestas, (***E = {{1,3},{2,3},{3,4},{3,5},{6,6}}***), representadas pelas linhas de ***a*** a ***d***. Nesta mesma figura, pode ser visto, também, que um vértice pode estar ligado a ele mesmo (ALOISE, 2008).

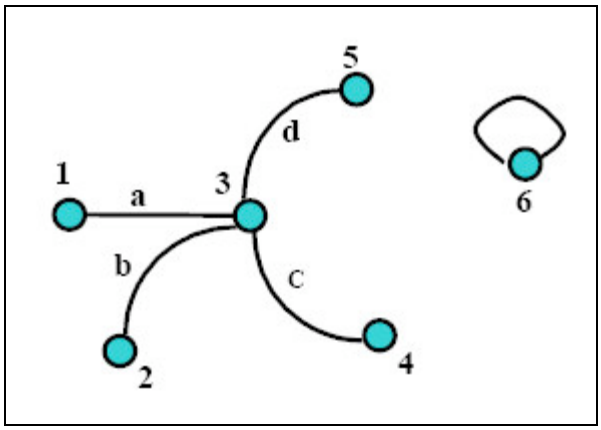

Figura 10. Exemplo de Grafos.

Fonte: ALOISE, 2008.

O número de vértices de um grafo determina sua classificação K_n, representado com ***1, 2, 3 ,4....n*** vértices, como pode ser visto na Figura 11.

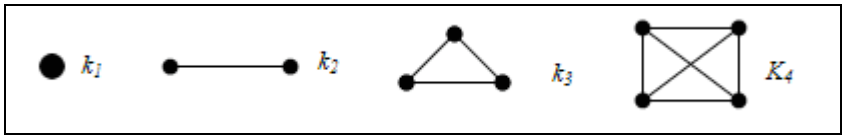

Figura 11. Exemplos de grafos K_1, K_2, K_3 e K_4.

Os grafos podem ser classificados de diversas maneiras e existem vários algoritmos para resolvê-los. Para o problema em questão, os grafos planares se encaixam nas características do desenho de uma planta baixa.

3.1.1 Grafos Planares

Segundo Boaventura (2006, p. 237), os grafos planares estão intimamente relacionados com a noção de mapa, como pode ser visto na Figura 12. Todos os vértices estão dispostos de uma forma que suas arestas não chegam a se cruzar, lembrando assim, um mapa. Da mesma forma que um mapa, uma planta baixa não terá arestas se cruzando, demonstrando, assim, a possibilidade de sua utilização no problema proposto. Esta característica também é análoga a um polígono simples. Um problema muito conhecido em que são aplicados grafos planares é o problema das 4 cores. Nele, um mapa pode ser colorido com quatro cores sem repeti-las entre as regiões adjacentes, isso quer dizer, que possibilita a separação das regiões.

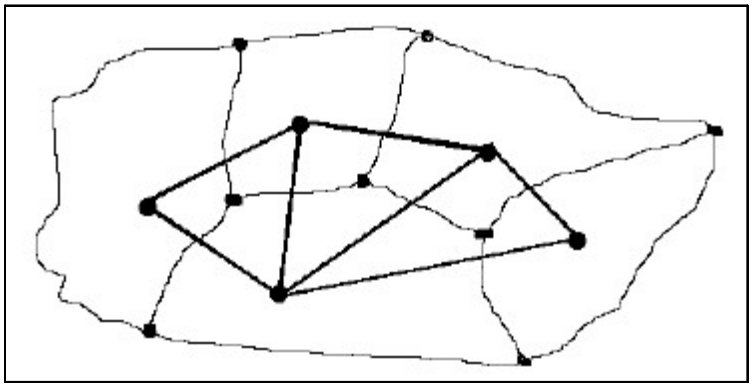

Figura 12. Grafo Planar com 5 regiões.
Fonte: ALOISE, 2008.

Como já descrito, a definição de um grafo planar depende se seus vértices e arestas permitirem sua reprodução num plano sem que as arestas se cruzem. Mas esta regra não pode determinar se um grafo não é planar por causa de uma de suas representações. Por exemplo, um grafo K_4 pode ser representado de diversas formas. Veja que na Figura 13, que apresenta algumas destas formas, existem dois desenhos onde as arestas se cruzam. Nas outras representações, o posicionamento diferente dos vértices modificou o desenho das arestas de tal forma que ficassem planares. Isso faz-se concluir que o grafo K_4 é planar.

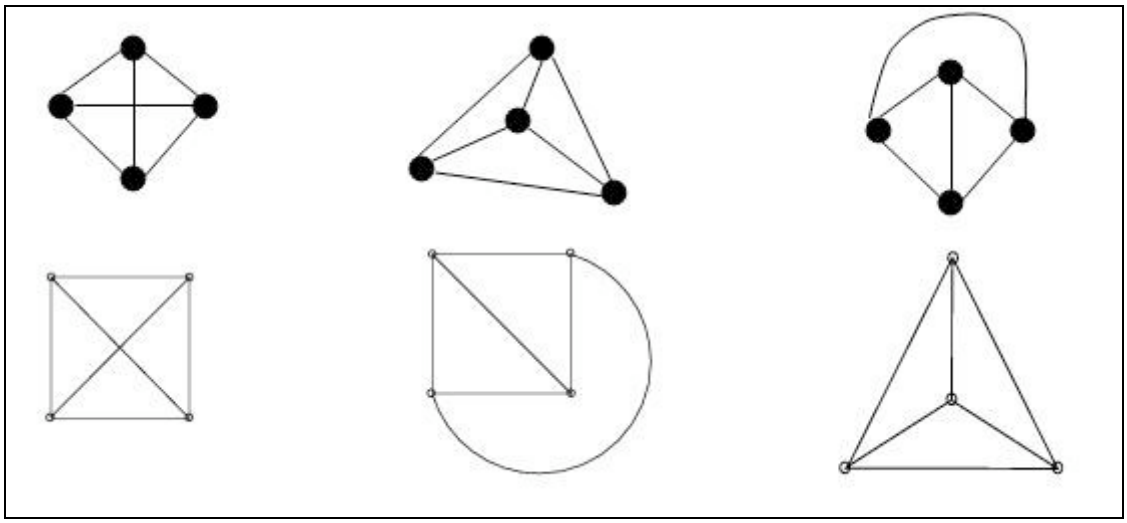

Figura 13. Exemplos de representações de grafos K_4.
Fonte: FERNANDES, 2009.

A pergunta que pode surgir com esta constatação é: “Mas então, poucos grafos não são planares?”. Para responder está pergunta foram criados teoremas e proposições para identificação de grafos não planares, mas não afirma que um grafo é planar.

Algumas proposições (FERNANDES, 2009):

Todos os grafos com classificação K_4 são planares;

Todo o subgrafo de uma grafo planar é planar;

Um grafo é planar se, e somente se, todos os seus componentes conexos são planares;

Um grafo é planar se, e somente se, qualquer subdivisão é um grafo planar;

Os grafos com classificação K_5 e $K_{3,3}$, Figura 14, não são planares;

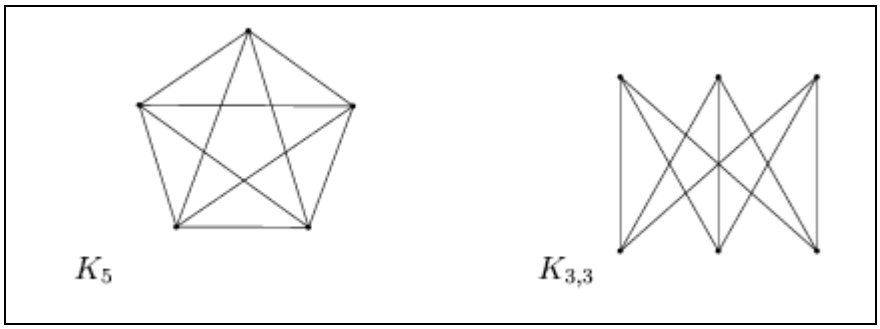

Figura 14. Exemplos de representações de grafos K_5 e $K_{3,3}$.
Fonte: FERNANDES, 2009.

Segundo Fernandes (2009), é possível criar uma contradição dizendo que o grafo $K_{3,3}$ é planar, $K_{3,3}$ tem comprimento ***6 (u, v, w, x, y, z)*** e pode ser desenhado com um hexágono, conforme Figura 15. Na Fig. A, todas as arestas foram ligadas totalmente contidas dentro do hexágono. A fim de evitar o cruzamento pode-se aplicar a seguinte estratégia: utilizar a aresta ***{z, w}*** totalmente contida, depois, para evitar o cruzamento com a aresta ***{z, w}***, a aresta ***{u, x}*** ficará totalmente fora do hexágono, desta maneira, obtém-se o resultado visto na Fig. B. "Como será possível desenhar a aresta ***{v, y}***?". Está é a contradição que confirma que o grafo $K_{3,3}$ não pode ser planar.

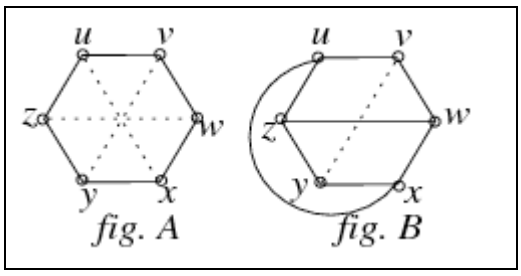

Figura 15. Grafo $K_{3,3}$ com comprimento 6.
Fonte: FERNANDES, 2009.

Os grafos K_5 e $K_{3,3}$ são mais que simples exemplos de grafos não planares, segundo o teorema de Kuratowski, 1930 citado em Diestel (2005):

> Um grafo não é planar se e só se contém um subgrafo homeomorfo a K_5 ou a $K_{3,3}$. Dois grafos são homeomorfos se um pode ser obtido por outro através da modificação de vértices.

Kuratowski, matemático polaco, apresentou este teorema em 1930 com o objetivo de fornecer a possibilidade de apontar a planaridade de um grafo sem depender da sua representação gráfica. No entanto, na prática, seu teorema torna-se complexo em sua execução, considerado um tempo útil devido à necessidade de avaliar se todos os subgrafos são homeomorfos a K_5 ou a $K_{3,3}$ (DIESTEL, 2005).

Outra forma de identificação de grafos planares é através da relação entre o número de arestas e vértices. Leonhard Euler, um matemático Suiço do século XVIII, fez uma descoberta através da percepção de que um grafo simples, com seu desenho planar conexo, divide o grafo em regiões fechadas. Neste momento, Euler fez uma relação entre o número de arestas, número de vértices e o número de regiões com a seguinte expressão:

$n - a + r = 2$ **se** $n \geq 3$ **então** $a \leq 3n - 6$

n – número de vértices ***a – número de arestas*** ***r – número de regiões***

Pode-se considerar como exemplo o grafo K_1 (Figura 11):

$n = 1 \quad a = 0 \quad r = 1 \qquad 1 - 0 + 1 = 2$

Para a condição de $n \geq 3$ pode-se utilizar um grafo K_5 (Figura 14)

$n = 5 \quad a = 10 \qquad 10 \leq 3*5 - 4$

Veja que a expressão se aplica a ambos os exemplos.

A partir destes estudos foi possível identificar que os grafos possibilitam o armazenamento e o caminhamento a partir das informações fornecidas pelo desenho. Porém, é necessário lembrar que o grafo trabalha apenas sob os dados já mapeados. Para realizar este mapeamento, é necessário utilizar a geometria computacional, já que este mapeamento pode ser realizado através dela. Os estudos sobre essa teoria serão apresentados na próxima seção.

3.2 Geometria Computacional (GC)

A geometria computacional oferece diversos algoritmos para solucionar problemas relacionados com formas geométricas no computador. Além disso, ela utiliza a análise de estruturas de dados específicas para melhor representar desenhos geométricos: polígono, retângulo, pontos, entre outros.

A aplicação proposta aqui dependerá da facilidade de armazenamento e recuperação das informações das formas geométricas de uma planta baixa.

Exemplo de representação de estruturas geométricas:

Ponto/Vértice: a estrutura de um ponto, ou vértice no caso de polígonos, pode ser representada por ***X*** e ***Y***; duas coordenadas para o ponto;

Reta, segmento de reta ou aresta: a estrutura de um segmento de reta ou aresta no caso de polígonos é composta da ligação de dois pontos;

Retângulo: é representado por quatro coordenadas: ***X1***, ***Y1*** e ***X2***, ***Y2***. A primeira dupla de coordenadas representa o ponto ***X***, ***Y*** do canto superior esquerdo e a segunda dupla representa o canto inferior direito, conseguindo, desta forma, representar um retângulo quando ocorrer as junções das retas;

Polígonos: são arestas que apresentam conexão somente através de seus vértices. Possuem uma característica plana e são segmentos fechados. O polígono é a principal forma geométrica utilizada neste trabalho. Sua estrutura é composta por diversos elementos destacado na Figura 16 (POLÍGONO, 2009):

Lados ou faces: segmentos de reta que une dois vértices ($\overline{AB}$, $\overline{BC}$, $\overline{CD}$, $\overline{DE}$, $\overline{EA}$);

Vértices: ponto de encontro de dois lados consecutivos (***A, B, C, D, E***);

Diagonais: segmentos de reta que unem dois vértices não consecutivos ($\overline{AC}$, $\overline{AD}$, $\overline{BD}$, $\overline{BE}$, $\overline{CE}$);

Ângulos internos: ângulos formados por dois vértices consecutivos (***a, b, c, d, e***);

Ângulos externos: ângulos formados pelo lado e o segmento de reta consecutivo a ele (***a*₁, *b*₁, *c*₁, *d*₁, *e*₁***).

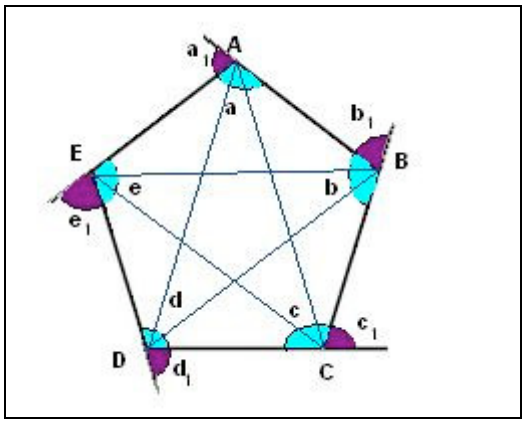

Figura 16. Elementos de um polígono.
Fonte: POLÍGONO, 2009.

Poliedro: também será encontrado nas estruturas de uma planta baixa, mas com uma característica 2D plana. O poliedro é um sólido geométrico composto por faces, sendo que cada face resulta num polígono. Seus principais componentes são vértices, arestas e faces (POLIEDRO, 2009).

Segundo Figueiredo (2005), a geometria computacional tem a seguinte definição:

> Geometria Computacional é o estudo sistemático de algoritmos eficientes para problemas geométricos.

As principais características de um problema geométrico são: entrada de dados; definição dos objetos geométricos; a forma de solução. Na Figura 17 são exibidos vários exemplos de problemas geométricos.

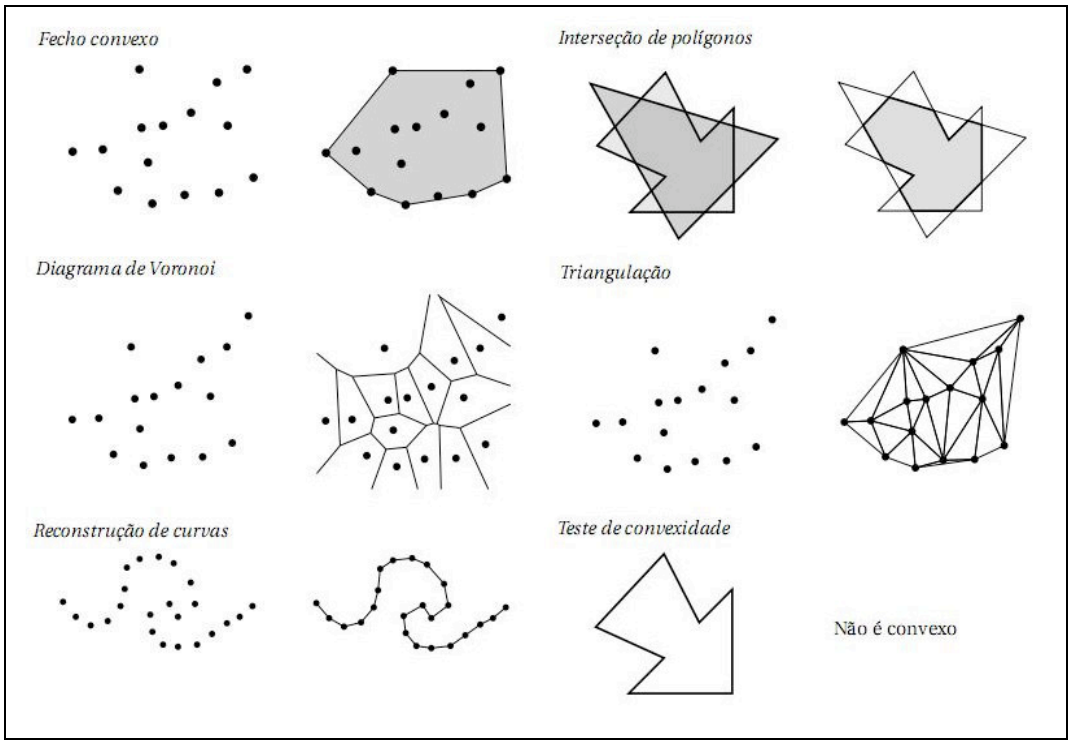

Figura 17. Exemplos de problemas Geométricos.
Fonte: FIGUEIREDO, 2005.

Os estudos sobre grafos e geometria apontaram uma complexidade inesperada no escopo do trabalho. Com a pesquisa sobre geometria computacional foram encontradas, diversas vezes, a afirmação de que existem problemas com a robustez numérica e a dificuldade de implementação de algoritmos. Ao mesmo tempo, foi encontrado um projeto *Open Source* chamado CGAL que apresentou características e funcionalidades muito parecidas com as necessidades deste trabalho.

3.3 Biblioteca de Algoritmos de Geometria Computacional (CGAL)

Nesta parte do trabalho serão vistas algumas características, objetivos e funcionalidades da CGAL, que é uma biblioteca de estrutura de dados e algoritmos de geometria computacional. Ela é mantida por vários colaboradores, mas surgiu de um consórcio criado por 8 instituições (TEILLAUD, 2009), Figura 18:

ETH Zurich (Suíça);

FU Free Universität – Berlin (Alemanha);

INRIA Sophia-Antipolis (França);

MLU Martin-LutherUniversität Halle – Wittenberg (Alemanha);

MPI Max-Planck Institut für Informatik – Saarbrücken (Alemanha);

RISC Linz (Austria);

Tel Aviv University (Israel);

Utrecht University (Holanda).

Figura 18. Instituições participantes do projeto CGAL.
Fonte: TEILLAUD, 2009.

O seu desenvolvimento iniciou em 1995 com o objetivo de implementar algoritmos geométricos básicos para facilitar sua utilização em aplicações complexas. Em 2003 foi fundada a companhia *Geometry Factory*, que realiza a comercialização de licenças, criação de componentes, suporte, desenvolvimento customizado e treinamentos. Outra iniciativa foi abrir o código e aceitar contribuições que são avaliadas pelo *Editorial Board* da CGAL.

Segundo CGAL *Editorial Board* (2009), a declaração da missão do CGAL é

> Make the large body of geometric algorithms developed in the field of computational geometry available for industrial applications (1996)[1].

Abaixo algumas estatísticas do CGAL (PIERRE, 2008):

Criado em C++ com mais de 500000 linhas de código;

10000 downloads por ano;

3500 páginas de manual, tornando-se sua principal fonte de referência;

Fórum discussão especializado, com participação de seus desenvolvedores;

100 packages;

20 desenvolvedores ativos;

2 tipos de licença, *Open Source* ou comercial;

Mais de 60 usuários comerciais; ver Figura 19.

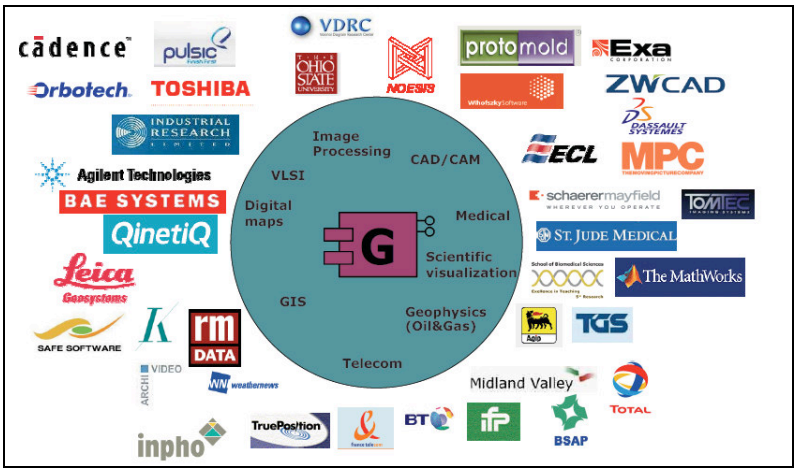

Figura 19. Usuários comerciais.

Fonte: PIERRE (2008).

1 Criar grande quantidade de algoritmos geométricos no campo da Geometria Computacional disponibilizando para aplicações industriais (1996).

3.3.1 Pacotes da CGAL utilizados neste Trabalho

Foram encontrados muitos materiais sobre CGAL e, quase todos são fornecidos no próprio site da biblioteca (CGAL, 2009). Outro ponto interessante é que, no fórum de discussão, os desenvolvedores e grande parte dos citados neste trabalho respondem perguntas a todo o momento, ou seja, participam continuadamente nas melhorias do projeto. Há alguns anos, os principais desenvolvedores da *Geometry Factory* ministram um curso no congresso SIGGRAPH da ACM (2009, http://www.siggraph.org/asia2009/).

A seguir, serão descritos alguns pacotes (*packeges*) da CGAL, destacando os que foram utilizados. A CGAL encontra-se na versão 3.5 e possui 16 grupos de pacotes, sendo que para apresentação do método foram necessários somente os pacotes 2D pela característica do desenho da planta baixa:

***Arithmetic and Algebra*:** pacotes com classes para métodos aritméticos e algébricos, como, por exemplo, conceitos para polinômios multivariados:

Algebric Foundations;

Number Type;

Polynomial;

Modular Arithmetic.

Combinatorial Algorithms: pacotes com algoritmos combinatórios como, por exemplo, o cálculo de todos os vizinhos mais distantes para os vértices de um polígono convexo:

Monotone and Sorted Matix Search;

Linear and Programming Quadratic Solver.

Geometry Kernels: muito importante, o pacote *2D and 3D Linear Geometry Kernel* traz todos os objetos (Figura 20) necessários aqui, além de predicados e operações para sua manipulação:

2D and 3D Linear Geometry Kernel;

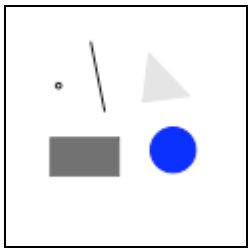

Figura 20. *2D and 3D Linear Geometry Kernel.*
Fonte: CGAL, 2009.

dD Geometry Kernel;

2D Circular Geometry Kernel;

3D Spherical Geometry Kernel.

Convex Hull Algoritms: os algoritmos para resolução do envoltório convexo são facilmente encontrados quando se busca informações sobre geometria computacional. Este algoritmo, conforme exemplo da Figura 21, identifica quais pontos formam um polígono convexo sem deixar nenhum ponto externamente isolado:

2D Convex Hull and Extreme Points;

3D Convex Hull;

dD Convex Hull;

Delaunay Triangulation.

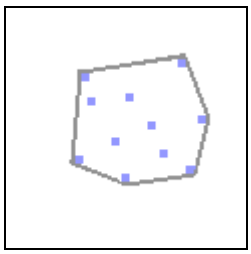

Figura 21. *Convex Hull*, problema muito conhecido na geometria computacional.
Fonte: CGAL, 2009.

Polygons: pacotes com classes para manipulação de polígonos, foram utilizados por apresentar predicados e operações sobre ponto e sequência de pontos, teste de convexidade entre outros, ver Figura 22:

2D Polygon (GIEZEMAN et al, 2009)*;*

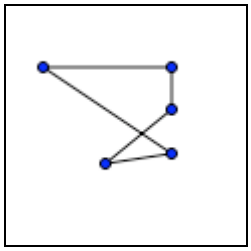

Figura 22. *2D Polygon.*
Fonte: CGAL, 2009.

2D Regularized Boolean Set-Operations

2D Boolean Operations on Nef Polygons;

2D Boolean Operations on Nef Polygons Embedded on the Sphere;

2D Polygon Partitioning;

2D Straight Skeleton and Polygon Offsetting

2D Minkowski Sums;

Polyhedra: pacotes com classes para manipulação de poliedros (superfícies com três dimensões: vértices, arestas e faces). O pacote mais importante deste conjunto é a *Halfedge*, que é uma estrutura de dados capaz de manter as informações de incidência de vértices, arestas e faces, como, por exemplo, mapas planares, ver Figura 23:

3D Polyhedral Surface;

Halfedge Data Structures (KETTNER et al, 2009)*;*

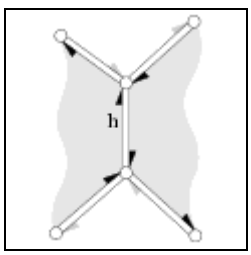

Figura 23. *Halfedge Data Structures.*
Fonte: CGAL, 2009.

3D Boolean Operations on Nef Polyhedra;

Convex Decomposition of Polyhedra

3D Minkowski Sum of Polyhedra.

Arrangements: pacotes que fazem, basicamente, o papel de um grafo planar. Responsável por arranjos planares, possui estruturas de dados que, junto com a *Halfedge*, permite a varredura dos vértices, função necessária para o mapeamento do desenho. O pacote *2D Arrangement* (WEIN et al, 2009) possui vários predicados neste sentido, ver Figura 24:

2D Arrangement;

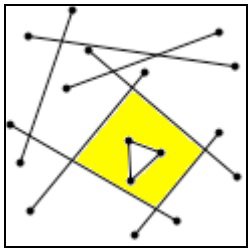

Figura 24. *2D Arrangement e 2d Intersection of Curve.*
Fonte: CGAL, 2009.

2D Intersection of Curves;

2D Snap Rounding;

2D Envelopes;

3D Envelopes.

Triangulations and Delaunay Triangulations: pacotes com classes para manipulação de triangulações, destacando-se a triagulação de *Delaunay*:

2D Triangulation 2D Triangulation Data Structure;

3D Triangulations;

3D Triangulation Data Structure;

3D Periodic Triangulations;

2D Alpha Shapes;

3D Alpha Shapes.

Voronoi Diagrams: pacotes com classes para manipulação de diagramas de *Voronoi*:

2D Segment Delaunay Graphs;

2D Apollonius Graphs (Delaunay Graphs of Disks);

2D Voronoi Diagram Adaptor.

Mesh Generation: pacotes com classes para definição e reconstrução de superfícies (malhas):

2D Conforming Triangulations and Meshes

3D Surface Mesh Generation

Surface Reconstruction from Point Sets

3D Skin Surface Meshing

3D Mesh Generation.

Geometry Processing: pacotes com classes para processamento geométrico, como, por exemplo, separação e triangulação de superfícies:

3D Surface Subdivision Methods

Triangulated Surface Mesh Simplification

Planar Parameterization of Triangulated Surface Meshes

2D Placement of Streamlines

Approximation of Ridges and Umbilics on Triangulated Surface Meshes

Estimation of Local Differential Properties

Point Set Processing.

Spatial Searching and Sorting: pacotes com classes para pesquisa e classificação espacial como, por exemplo, apontar a vizinhança de um vértice:

2D Range and Neighbor Search;

Interval Skip List;

dD Spatial Searching;

dD Range and Segment Trees;

Intersecting Sequences of dD Iso-oriented Boxes;

AABB Tree;

Spatial Sorting.

Geometric Optimization: pacotes com classes para otimização geométrica como, por exemplo, calcular a distância de um ponto interno com o envoltório convexo:

Bounding Volumes;

Inscribed Areas, Optimal Distances;

Principal Component Analysis.

Interpolation: implementa diferentes métodos de interpolação de dados dispersos.

2D and Surface Function Interpolation.

Kinetic Data Structures: facilita a execução e depuração de estruturas de dados cinéticos:

Kinetic Data Structures;

Kinetic Framework.

Support Library: pacotes de aplicativos e bibliotecas que adicionam mais funcionalidades ao CGAL, como: interface, visualização de resultados e entrada e saída:

STL Extensions for CGAL;

CGAL and the Boost Graph Library;

CGAL and Boost Property Maps;

Handles and Circulators;

Geometric Object Generators;

Profiling tools, Hash Map, Union-find, Modifier;

IO Streams;

Geomview;

CGAL and the Qt Graphics View Framework;

CGAL Ipelets.

3.3.2 Estrutura

A CGAL é dividida em três partes principais, conforme Figura 25.

Kernel: o *Kernel* contém objetos geométricos básicos chamados de *Kernel Geometry* como, por exemplo, os pontos, vetores e linhas. Além dos objetos, oferece predicados para verificação da sua posição e operações como interseção e cálculo da distância. Estas operações são chamadas de *Predicates and Constructions*. O *Kernel* tem uma representação própria utilizando a programação genérica.

Biblioteca básica: nesta parte são encontrado um conjunto de algoritmos e estruturas de dados como, por exemplo, o envoltório convexo mais conhecido como *convex hull*, Triangulações (Delaunay), mapas planares (*planar maps*), arranjos (*arrangement*), superfície do poliedro (*polyedra surface*), menor círculo envolvente (*smallest enclosing circle*) e estruturas de consultas multidimensional. A envoltória convexa é o exemplo mais comum para explicar os problemas da programação de algoritmos geométricos utilizando precisão de ponto flutuante. Um destes exemplos pode ser visto em Hanniel (2000).

Biblioteca de suporte: na biblioteca de suporte são encontrados outros pacotes que servem para interface, entrada e saída, tipos numéricos e outras facilidades.

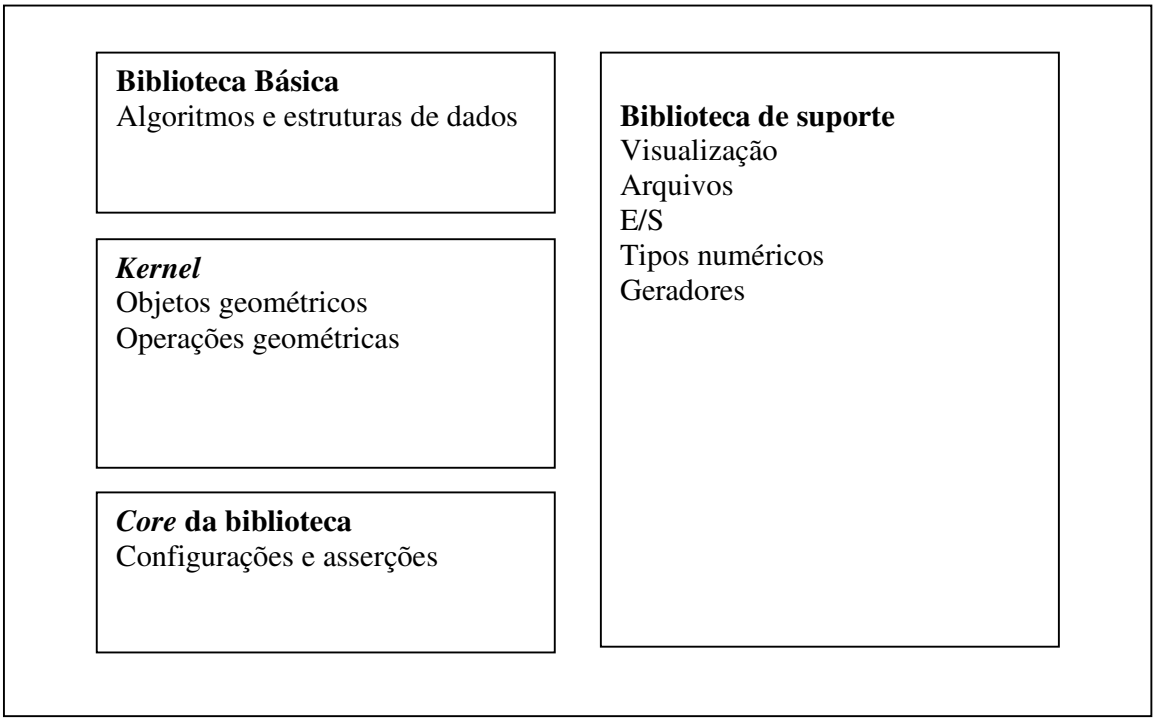

Figura 25. Estrutura do CGAL.

Fonte: PIERRE, 2008.

Uma das grandes facilidades que o CGAL disponibiliza dentro de sua estrutura é o seu *Kernel*, que traz vários predicados e operações já construídas. Na Figura 26, pode ser visto o predicado de orientação que determina se o ponto **r** está à esquerda, direita ou colinear aos pontos ***q*** e ***p***. No desenho da circunferência, o predicado *in_circle* verifica a posição do ponto ***s*** no círculo, ou seja, se o ponto está dentro, fora ou sobre ao círculo,

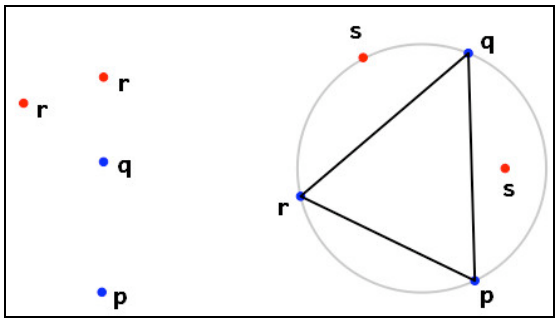

Figura 26. Orientação de pontos e verificação da localização de um ponto numa circunferência.

Fonte: PIERRE, 2008.

Na teoria, a implementação destes predicados oferece diversos métodos, sejam eles geométricos ou matemáticos. Estes sempre apresentam grande complexidade e robustez numérica. No caso da orientação pode ser citado o produto cruzado como método para

determinar se segmentos consecutivos se dirigem para a esquerda ou para direita. Isso é muito importante para diversas manipulações da estrutura de dados geométricas. O produto cruzado, ver Modelo 1, utiliza 3 pontos para saber a posição relativa do ponto ***r***. Se o resultado for igual a zero aponta que o ***r*** é colinear ao ponto ***p*** e ***q***, se o resultado for maior que zero está à direita e se for menor estará à esquerda do segmento ***pq***; ver Figura 27.

orientação(p,q,r) = sign((px-rx) * (qy-ry) - (py-ry) * (qx-rx))

Modelo 1. Calculo do produto cruzado para determinar a direção de um ponto.
Fonte: CORMEN, 2002.

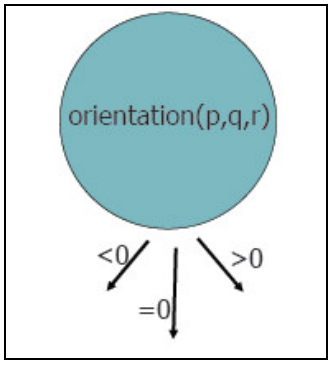

Figura 27. Resultado do ponto cruzado.
Fonte: PIERRE, 2008.

Já para exemplo das operações, a Figura 28 traz duas operações muito úteis para o desenvolvimento de algoritmos geométricos. Dado dois segmentos de reta, verificar se existe interseção e, se sim, em que ponto. Outro exemplo da figura aponta o centro do círculo com a informação de 3 pontos. Esta operação lembra a utilização de um compasso.

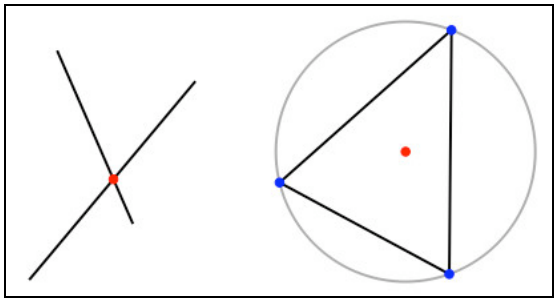

Figura 28. Interseção e centro do círculo.
Fonte: PIERRE, 2008.

3.3.3 Programação Genérica e CGAL

Como visto, a estrutura do CGAL traz contribuições importantes para alavancar o desenvolvimento de soluções que venham a utilizar a geometria computacional. Outro ponto forte do CGAL é a utilização da programação genérica. A programação genérica foi implementada, inicialmente, na S*tandart Template Library* (STL) do C++. Ela traz a idéia da utilização de *templates*, *namespaces* e *iterators* resultando em grande eficiência.

Os pontos chaves da programação genérica são descritos por Poin (2009):

O algoritmo possui pressupostos mínimos sobre a abstração de dados facilitando sua manipulação, ou seja, uma estrutura de dados pode ser declarada durante o algoritmo sem obedecer a um tipo fixo de dado;

Eleva um algoritmo de concreto para um nível generalizado sem perder a eficiência, ou seja, a forma mais abstrata, tal que, quando voltar a ser especificado o caso concreto, o resultado é tão eficiente quanto o algoritmo original;

Oferecer mais do que um algoritmo genérico para a mesma finalidade e em mesmo nível de abstração. Isso introduz a necessidade de prever, de forma precisa, as características do domínio para o qual cada algoritmo é mais eficiente.

Com o auxílio da programação genérica, o objetivo do CGAL é trazer robustez, eficiência, flexibilidade e facilidade em sua utilização.

3.3.4 A Estrutura de Dados *Halfedge*

A *Halfedge data structure* é uma combinação de estrutura de dados, interpretações geométricas e classes responsáveis pela sua manipulação (CGAL, 2009). Com o mapeamento do desenho, a *halfedge* oferece as informações sobre os vértices (pontos), arestas (segmentos de reta), faces e uma relação de incidência entre eles, como pode ser visto na Figura 29.

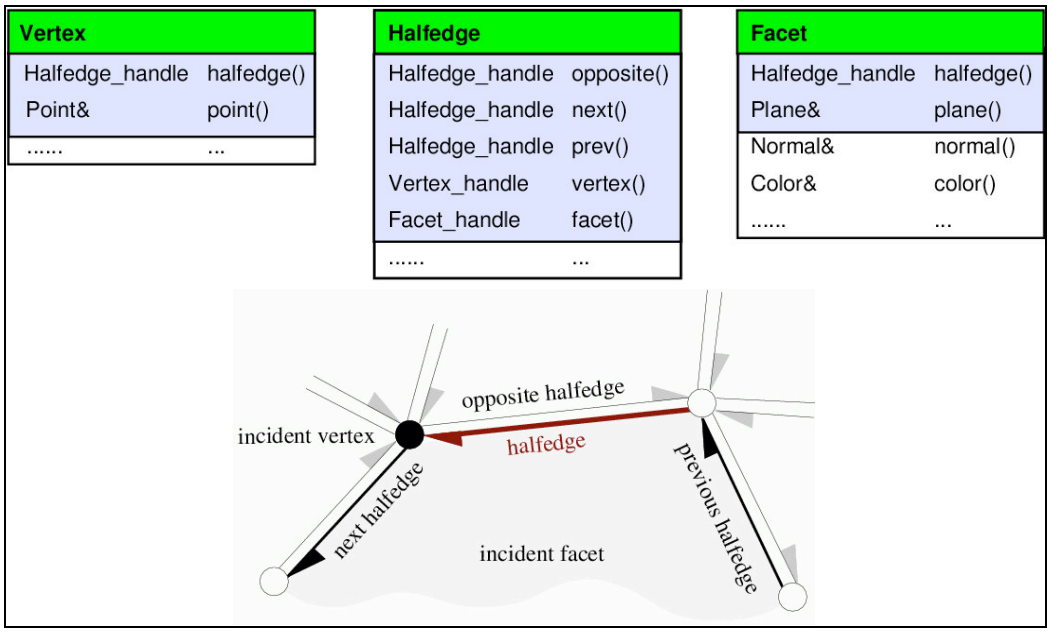

Figura 29. As três classes que formam a estrutura *halfedge*: *vertex*, *halfedge* e *face*.
Fonte: CGAL, 2009.

3.3.5 Operações utilizando a *Halfedge*

Nos exemplos a seguir, é utilizado o poliedro 3D para ajudar na compreensão do funcionamento do *iterator*. Neste trabalho, porém, será utilizado o poliedro 2D e *Arrangement* por causa da característica plana, que será melhor detalhado na próxima seção. O *iterator* é como um ponteiro que determina um local na estrutura. Ele tem a finalidade de apontar um único ou uma sequência de locais.

Iterar sobre todos os vértices: no código abaixo é criado um *iterator* para os vértices. Com este *iterator* é possível percorrer todos os vértices, conforme exemplo da Figura 30.

```
Vertex_iterator v;

for( v = polyhedron.vertices_begin(); v != polyhedron.vertices_end(); ++v ){

        // Pode manipular o v conforme necessário.

}
```

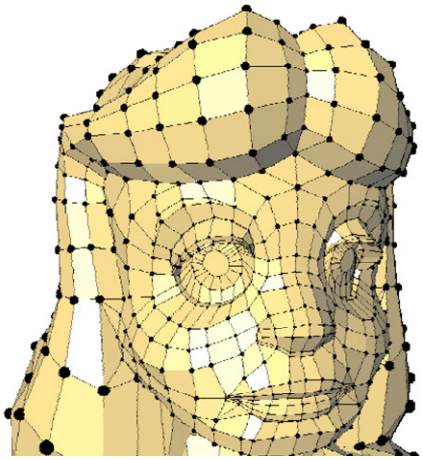

Figura 30. Exemplo de iteração através de todos os vértices.

Fonte: PIERRE, 2008.

Circular em volta das faces: no código abaixo é criado um *iterator* para circular as faces da *Halfedge*. Com este *iterator* é possível percorrer todos os vértices de uma face, conforme exemplo da Figura 31.

```
Halfedge_around_facet_circulator he,end;
he = end = f->facet_begin();
CGAL_For_all(he,end){
        // He pode ser manipulado
}
```

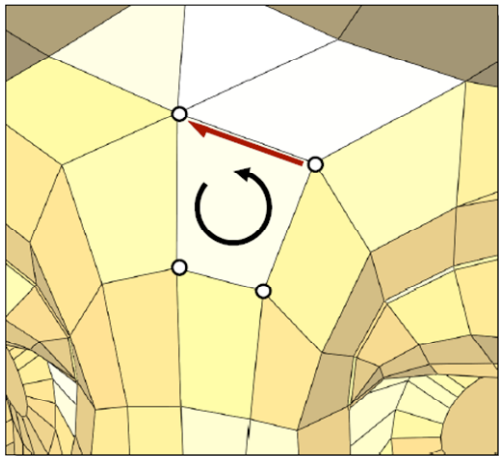

Figura 31. Circular em volta das faces.

Fonte: PIERRE, 2008.

Circular em volta dos vértices: no código abaixo é criado um *iterator* para circular um vértice da *Halfedge*. Com este *iterator* é possível percorrer todas as faces de um vértice, conforme exemplo da Figura 32.

```
Halfedge_around_vertex_circulator he,end;
he = end = f->vertex_begin();
CGAL_For_all(he,end){
        // He pode ser manipulado
}
```

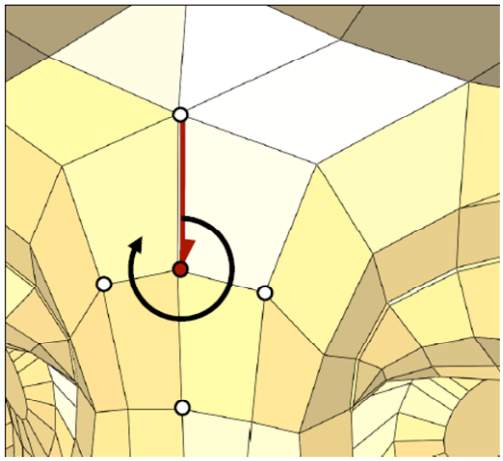

Figura 32. Circular em volta de um vértice.
Fonte: PIERRE, 2008.

3.3.6 *Arrangements*

O pacote de algoritmos da CGAL, chamado de *Arrangements*, traz soluções para arranjos e curvas de até duas dimensões, ou seja, formas geométricas planas.

Segundo Wein et al (2009) dado um conjunto ***C*** de curvas planas, o *arrangement* ***A(C)*** é a subdivisão do plano em partes de zero, uma ou duas dimensões, estruturas aqui já citadas chamadas vértices, arestas e faces, respectivamente induzidas pelas curvas em ***C***. A explicação de sua estrutura assemelha-se muito aos mapas planares. A diferença é que os

arrangements são considerados um conjunto de curvas e, em alguns casos, permitem interseção. Sua representação básica é dada pela estrutura de dados chamada de DCEL (*Doubly Connected Edge List*).

Segundo Wein et al (2009, p.1644, tradução nossa) a idéia da estrutura DCEL é

> Representar cada aresta usando um par de *halfedges* dirigido, uma indo da *xy-lexicographically* menor (à esquerda) da extremidade da curva, em direção a sua *xy-lexicographically* maior (direita) da extremidade, e o outro, conhecido como seu *halfedge* gêmea, indo na direção oposta. Como cada *halfedge* é dirigida, dizemos que ela tem um vértice de origem e um vértice de destino. *Halfedges* são usadas para separar faces, e para conectar vértices (com exceção dos vértices isolados, que são sem conexão).

A Figura 33 traz um exemplo de um arranjo de segmentos interiores disjuntos com alguns registros da estrutura DCEL que representam isso. Segundo Wein et al (2009, p.1645, tradução nossa) a face ilimitada f_0 tem um único componente ligado que forma um buraco dentro dela, e este buraco é composto se várias faces. A *halfedge* e é dirigida a partir do seu vértice de origem v_1 até o seu vértice de destino v_2. Esta aresta, juntamente com sua gêmea e', correspondem a um segmento de reta que liga os pontos associados com v_1 e v_2 e separa a face f_1 de f_2. O e_{prev} e o e_{next} do e fazem parte da corrente que forma a borda externa da face f_2. A face f_1 tem uma estrutura mais complicada, porque contém dois buracos no seu interior: um buraco é composto por duas faces adjacentes f_3 e f_4, enquanto o outro buraco é composto de duas arestas. A face f_1 também contém dois vértices isolados u_1 e u_2 no seu interior. É possível perceber que uma *halfedge* é sempre seguida de outra, montando uma espécie de lista circular compartilhando um mesmo vértice, isto é, o vértice destino de uma será o vértice origem da outra. Para isso, como visto na seção anterior, cada *halfedge* armazena um ponteiro (*iterator*) que permite que seja percorrido entre eles, na direção anti-horária. Esta lista forma uma corrente de tal forma que todas as arestas de uma cadeia são incidentes a uma mesma face dentro de sua borda. Esta cadeia é chamada de CCB (*Connect Component of Boundary*).

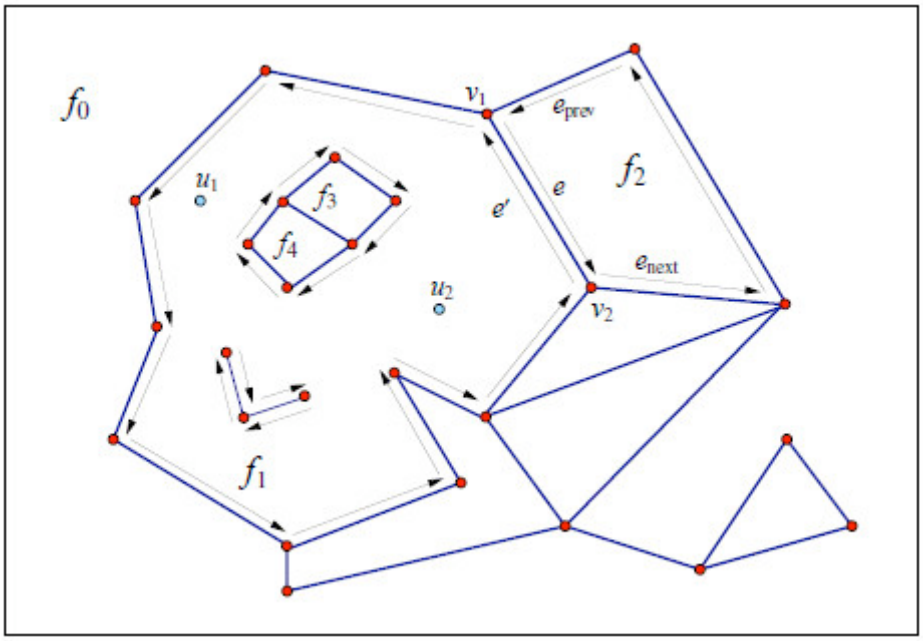

Figura 33. Exemplo da estrutura DCEL.

Fonte: WEIN et al, 2009, p.1645.

É possível comparar o exemplo da Figura 33 com um exemplo prático utilizando uma planta baixa com 5 cômodos, ver Figura 34. A face ilimitada f_0 seria o buraco principal, que é formado pelas faces f_1, f_2, f_3, f_4 e f_5. As setas são os *halfedges* que ligam os vértices formando uma borda externa identificando, assim, cada face (polígono ou cômodo) da planta. Esta estrutura, como pode ser visto, é basicamente um grafo planar. A CGAL afirma, muitas vezes, que o conjunto de curvas C está embarcado ou incorporado a um grafo planar. Como exemplo, Flato et al (2000) utilizam *arrangements* para uma aplicação de topologia de mapas.

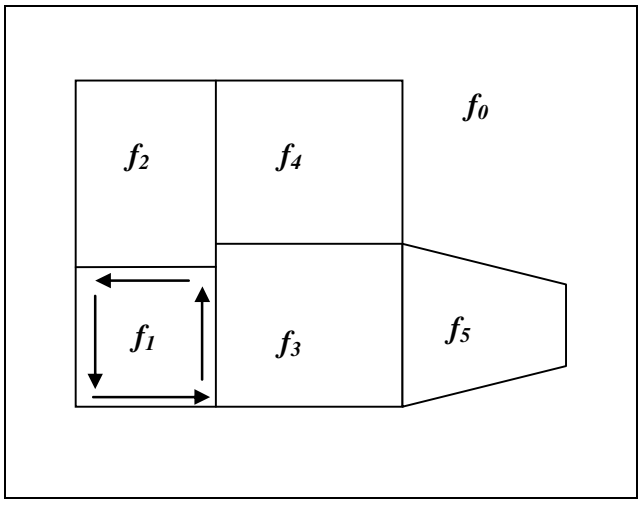

Figura 34. Exemplo da estrutura DCEL formando uma planta baixa.

3.4 Considerações

Os estudos realizados sobre desenhos geométricos demonstraram a possibilidade de mapear, armazenar e recuperar as informações dos desenhos da planta baixa de um forro. As técnicas de grafos e geometria computacional, trabalhando juntas, podem apresentar o resultado esperado, porém, a partir dos estudos realizados foi possível identificar uma preocupação com os cálculos geométricos no que se refere à capacidade numérica. Com isso, o estudo de uma biblioteca para resolução de algoritmos geométricos que apresenta a ligação destas duas técnicas foi encontrado e realizado. O próximo capítulo apresenta o corte, que utilizará as informações obtidas pelo desenho geométrico e determinará os itens e o aproveitamento dos materiais necessários para o projeto.

4 PROBLEMA DO CORTE (PC)

Este capítulo tem por objetivo descrever o problema do corte, sua classificação e tipologia. Como visto na introdução, o problema do corte consiste em cortar um objeto ao longo de seu comprimento em itens (pedaços) de comprimentos especificados (provavelmente menores) com cada um tendo um valor de utilidade.

Segundo Arenales et al (2007), na indústria, vários processos exigem que itens sejam produzidos a partir do corte de peças maiores que estão em estoque ou são fabricadas. O corte pode ser apenas de uma dimensão (unidimensional) obedecendo somente a largura ou o comprimento da peça a ser cortada; por exemplo, cortar barras de aço, bobinas de papel, rolos de filme. Já no problema de duas dimensões; o corte pode ser realizado tanto na largura como no comprimento, por exemplo, cortar placas de madeira, tecido, chapas de aço e etc. Existe mais dimensões de corte que não serão abordados neste trabalho, que é o caso do corte de três dimensões, utilizado para cortar blocos de matéria-prima para colchões e travesseiros, ou então, criar uma espécie de corte de um container para determinar a melhor forma de se preencher os espaços num carregamento.

Com a diversidade de problemas que podem ser resolvidos com o corte e empacotamento, Dyckhoff (1992) criou uma estrutura que divide os problemas em dimensões espaciais e abstratas, a Figura 35 exibe esta estrutura selecionando exemplos conforme sua característica. Dyckhoff (1992) também criou uma forma de classificar estes problemas através da identificação de suas estruturas lógicas.

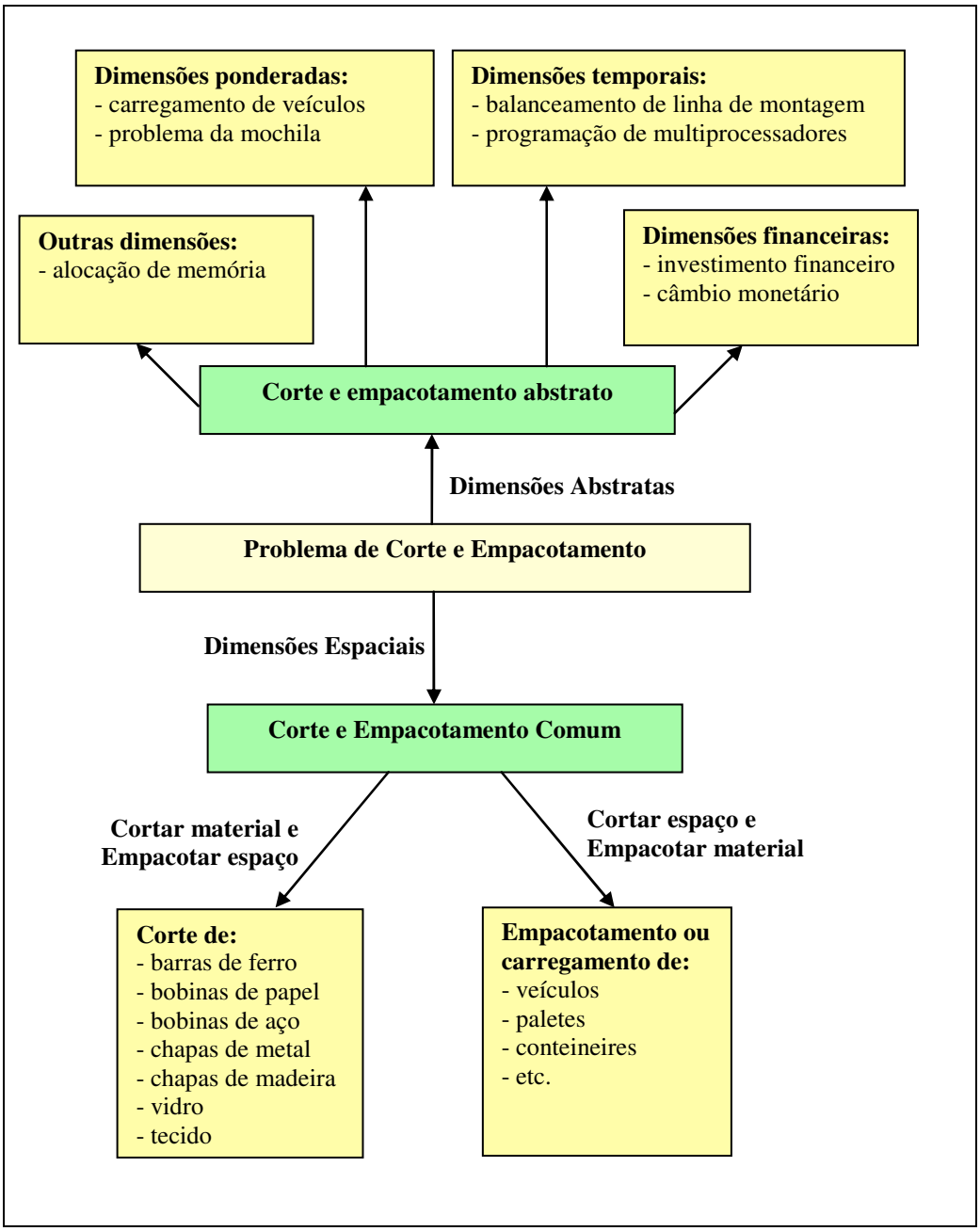

Figura 35. Estrutura dos problemas de corte e empacotamento.

Fonte: DYCKHOFF, 1992 apud VELASCO, 2005.

As características que montam esta estrutura lógica são classificadas da seguinte forma:

Dimensionalidade: como já citado, um problema pode ser classificado em diversas dimensões, unidimensional, bidimensional, tridimensional e N dimensional;

Medidas quantitativas: as variáveis que indicam o número de objetos e itens a serem cortados podem ser discretos: com medidas bem definidas (números naturais) ou contínuos: com medidas continuas (números reais);

Formas das figuras: o formato das figuras também é uma característica que pode determinar a complexidade da resolução do problema. As figuras se distinguem por sua forma podendo diferir em tamanho ou orientação. O tamanho pode ser medido por seu comprimento, área ou volume e sua orientação pode ser fixa ou permitir rotação;

Sortimento: os formatos das figuras em um problema apresentam uma grande diversidade, como objetos com formatos idênticos, por exemplo, painéis para paredes de divisórias e outros com formatos distintos como circuitos impressos;

Disponibilidade: três fatores são considerados a respeito da disponibilidade: o limite inferior, superior e quantidade podendo ser restrita ou irrestrita, a sequência, ordem e a data de utilização. Como exemplo de disponibilidade pode-se citar a utilização de uma demanda de estoque limitado para produção de um determinado padrão de corte;

Restrição de padrão: as restrições são sempre ligadas às características do problema, podendo ser geométricas e operacionais. Estas características são dividas em quatro grupos: o espaço entre os itens que pode ser considerado um espaço de perda, dependendo do problema e do material a ser cortado; a posição dos itens, por exemplo, carregamento de produtos frágeis; o tipo e número de cortes, podendo ser guilhotinado e ortogonais, oferecendo a possibilidade de padrões de corte;

Restrições de alocação: as restrições de alocação é uma propriedade fundamental em problemas típicos de corte que, no caso, serão utilizadas neste trabalho. Poderia ser apontado com exemplo um carregamento de pallet. Os números de estágios são importantes porque determinam a quantidade de passos para definição de um padrão, o número, freqüência ou sequência de padrões. A alocação pode ser dinâmica, dependendo da disponibilidade dos objetos podendo haver realocação ou estática com os itens previamente conhecidos;

Objetivos: significa a utilização de um critério a ser maximizado ou minimizado expressando, assim, a dimensão da eficácia da solução apontada. Neste trabalho, poderiam ser apontados diferentes objetivos como, por exemplo, maximizar o aproveitamento dos

itens a serem cortados ou produzidos, ou, então, minimizar a perda e o tempo de orçamento e montagem;

Estados da informação e variabilidade: determinar se os dados dos problemas são determinísticos ou estocásticos, ou, então, exatos ou variáveis. Por exemplo, o padrão apontado neste trabalho pode seguir uma determinada demanda e possuir passos determinísticos ou pode oferecer, após, finalização do projeto a possibilidade de variação determinada pelo cliente.

A Tabela 1 exibe a nomenclatura de Dyckhoff. Esta nomenclatura influencia, diretamente, na escolha e na complexidade do método de solução. A tipologia, como pode ser apresentada na tabela, leva em consideração quatro características, dimensionalidade, tipo de alocação, sortimento de objetos e sortimento de itens.

Tabela 1. Nomenclatura da tipologia.
Fonte: VELASCO, 2005.

Dimensionalidade	Tipo de Alocação
(1) Unidimensional; (2) Bidimensional; (3) Tridimensional; (N) N-dimensional, com N>3.	(B) Todos os objetos e uma seleção de itens; (V) Uma seleção de objetos e todos os itens.
Sortimento de Objetos	**Sortimento de Itens**
(O) Um objeto; (I) Objetos de formatos idênticos; (D) Objetos de formatos distintos.	(F) Poucos itens de diferentes formatos; (M) Muitos itens de muitos formatos distintos; (R) Muitos itens de formatos distintos em relativa quantidade; (C) Itens de formatos congruentes.

Segundo Velasco (2005), podem ser formados 96 tipos de problemas de PC e PCE, levando em consideração as quatro características em questão. Alguns destes tipos podem ser visualizados na Tabela 2.

Tabela 2. Tipologia de alguns problemas.

Fonte: VELASCO, 2005.

Problema	Tipo
Problema da mochila clássico	*1/B/O/*
Problema da mochila multidimensional	*/B/O/*
Problema do carregamento de *pallet*	*2/B/O/C*
Problema do carregamento de veículos	*1/V/I/F* ou *1/V/I/F*
Problema do carregamento de contêiner	*3/V/I/* ou *3/B/O/*
Problema do *bin packing* clássico	*1/V/I/M*
Problema do *bin packing* dual	*1/B/O/M*
Problema do *bin packing* bidimensional	*2/V/D/M*
Problema do *cutting stock* clássico	*1/V/I/R*
Problema do *cutting stock* bidimensional	*2/V/I/R*
Problema do *cutting stock* generalizado	*1/ / /*, *2/ / /* ou *3/ / /*
Problema do balanceamento de uma linha de montagem	*1/V/I/M*
Problema de alocação de memória	*1/V/I/M*
Problema de alocação de tarefas em multiprocessador	*1/V/I/M*
Problema de câmbio monetário	*1/B/O/R*
Problema de investimento financeiro em multiperíodos	*N/B/O/*

Neste trabalho será utilizado o PC, levando em consideração apenas uma dimensão. Inicialmente, seria utilizado, também, o problema de duas dimensões. Mas com os testes realizados, foi possível identificar que, como não serão tratadas plantas com curvaturas ou circunferências, o sortimento dos objetos será pequeno, podendo-se, então, dizer que o trabalho pode se encaixar em dois tipos de problemas: *bin packing* e *cutting stock* clássicos.

4.1.1 Problema do Corte Unidimensional (PCU)

A Figura 36 ilustra uma barra de comprimento 200 cm e uma solução, produzindo 2 frações de comprimento 33 cm, 1 pedaço de 40 cm e outro de 90 cm, resultando em uma perda de 4 cm. Aqui, os comprimentos das frações (33 cm, 40 cm e 90 cm) definem 3 tipos, que podem ser produzidos em quaisquer quantidades.

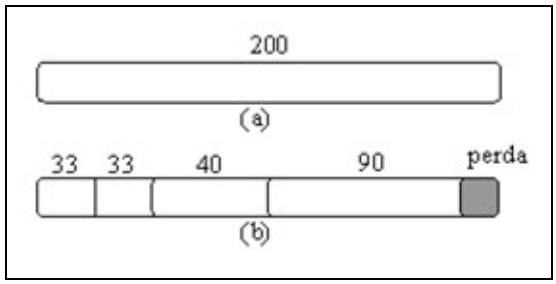

Figura 36. (a) barra a ser cortada. (b) uma solução factível.

Fonte: YANASSE, 1997.

Observa-se que o corte é feito em apenas uma dimensão do objeto, ou seja, somente obedecendo a largura ou o comprimento como já citado. Problemas com esta característica são chamados Problemas de Corte Unidimensional porque o item a ser cortado possui uma medida, comprimento ou largura comum ao objeto principal (YANASSE, 1997). O mesmo problema pode ser descrito como problema do empacotamento, sendo que os itens a serem produzidos podem ser empacotados para formação de um objeto padrão. Seria o mesmo que dizer que as frações (33 cm, 40 cm e 90 cm) foram empacotadas dentro da barra de 200 cm.

O problema é achar a melhor forma de se cortar o objeto para produzir os itens, de modo que, o valor de utilidade total seja máximo (equivale a dizer que a perda de material seja mínima).

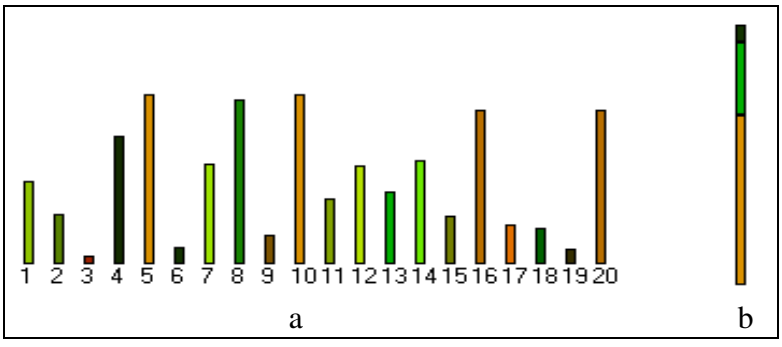

Figura 37. São 20 Pedaços(a) que devem produzir uma barra(b).
Foi produzida 1 barra utilizando os pedaços 5,13 e 19.

Observando o problema acima, percebe-se que ainda sobraram itens a serem utilizados, este problema, então, pode ser visualizado considerando um estoque, ou seja, determinado número de objetos que devem ser cortados para produção de itens menores necessários para suprir um estoque, ou, ainda, a utilização de um estoque de itens que deverão

produzir o menor número possível de objetos. Na Figura 38, é possível visualizar a diferença e o aumento da complexidade ao estender o problema a múltiplos objetos.

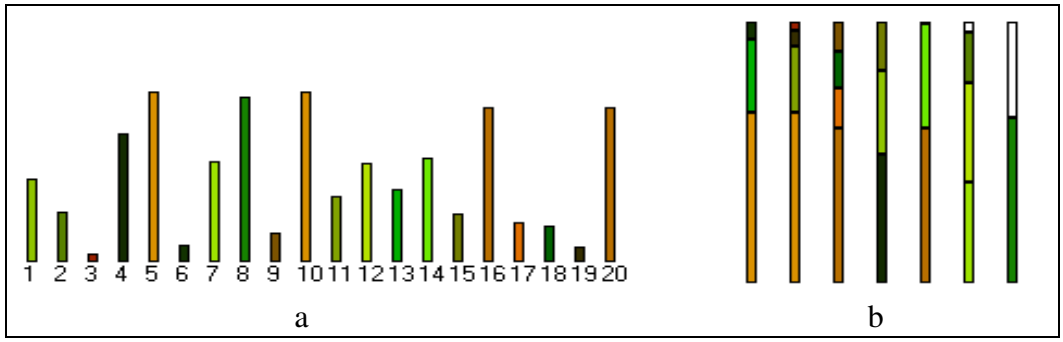

Figura 38. São 20 Pedaços(a) que devem produzir barras (b) com a menor perda possível. Foram produzidas 7 barras completas e 3 que tiveram perdas.

4.2 Modelos Matemáticos

Após a identificação, a definição e a descrição do PC é necessária a criação de um modelo matemático para que, a partir deste momento, seja possível avaliar e buscar possíveis algoritmos para resolvê-los.

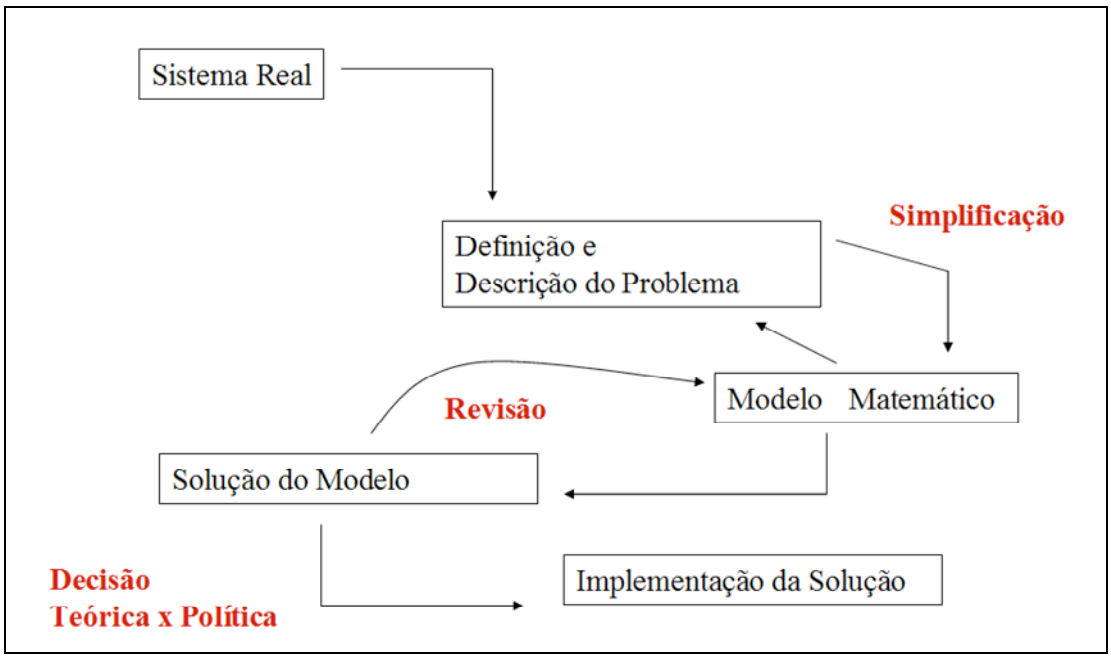

Figura 39. Processo de construção de um modelo matemático.

Fonte: RANGEL, 2009.

A Figura 39 demonstra o processo de construção de um modelo. Para a criação do modelo matemático é necessário fazer o levantamento de alguns elementos como:

Decisões: identificar as possíveis soluções (Definir Variáveis de Decisão);

Objetivos: definir critérios de avaliação capazes de indicar que uma decisão é preferível a outra (Definir Função Objetivo);

Restrições: identificar quais as restrições que limitam as decisões a serem tomadas (Definir Conjunto de Equações ou Inequações).

Dentre as classes de modelos de otimização, o PC se encaixa na classe dos problemas linear inteiro. Um exemplo clássico de um modelo de otimização. A forma geral do Modelo 2 será utilizada para modelagem matemática dos problemas já descritos.

min (ou max) **(função objetivo)**

sujeito a

(restrições principais - equações ou inequações)

(tipo das variáveis de decisão)

Modelo 2. Forma geral de um modelo de otimização.
Fonte: RANGEL, 2009.

4.2.1 Modelo Matemático do PCU

Dados do problema PCU:

m: número de tipos de itens;

vi: valor de utilidade do item tipo ***i, i =1, ..., m***;

li: comprimento do item tipo ***i, i =1, ..., m***;

***L*:** comprimento da barra.

Variáveis de decisão:

xi: quantidade produzida de itens do tipo ***i, i =1, ..., m***;

O problema pode ser formulado por:

$$\max imizar\ z = \sum_{i=1}^{m} v_i x_i \qquad (3.1)$$

$$sujeito\ a:$$

$$\sum_{i=1}^{m} l_i x_i \leq L \qquad (3.2)$$

$$x_i \geq 0\ e\ inteiro, i = 1,...,m. \qquad (3.3)$$

Modelo 3. Modelo genérico para o PCU.

O Modelo 3 possui uma restrição chamada restrição física (3.2). Esta restrição consiste em que o somatório dos itens utilizados não exceda o tamanho do objeto. Alguns problemas podem apresentar condições adicionais, como a quantidade limitada de itens, representada por $\boldsymbol{d_i, i = 1,...,m}$. Desta forma o modelo pode ser descrito por:

$$\max imizar\ z = \sum_{i=1}^{m} v_i x_i \qquad (4.1)$$

$$sujeito\ a:$$

$$\sum_{i=1}^{m} l_i x_i \leq L \qquad (4.2)$$

$$0 \leq x_i \leq d_i\ e\ inteiro, i = 1,...,m. \qquad (4.3)$$

Modelo 4. Problema do Corte Restrito.

O Modelo 4 apresenta uma restrição bastante utilizada em problemas que exigem corte em demanda. Em outras palavras, a maneira como um objeto em estoque é cortado para a produção de itens demandados.

Pode-se, ainda, ter o caso em que apenas uma unidade de um item possa ser utilizada. Desta forma, o modelo pode ser descrito por:

$$\max imizar\ z = \sum_{i=1}^{m} v_i x_i \qquad (5.1)$$

$$sujeito\ a:$$

$$\sum_{i=1}^{m} v_i x_i \leq L \qquad (5.2)$$

$$x_i = 0\ ou\ 1,\ i = 1,...,m. \qquad (5.3)$$

Modelo 5. Problema do corte de uma dimensão binário, ou seja, 0 ou 1.

O foco de Zonta (2005) foi resolver o PCU tomando como base o Modelo 5. O problema do Modelo 5 sofreu uma alteração na variável $\mathbf{v_i}$ a qual foi substituída por $\mathbf{l_i}$. Isto significa que o valor de utilidade de cada item será o seu próprio comprimento. Veja abaixo como ficou o modelo utilizado:

$$maximizar\ \ z = \sum_{i=1}^{m} l_i\ x_i \qquad (6.1)$$

$$sujeito\ a:$$

$$\sum l_i\ x_i \leq L \qquad (6.2)$$

$$x = 0\ ou\ 1,\ i = 1,...,m. \qquad (6.3)$$

Modelo 6. Problema do corte 0 ou 1.

Como o problema proposto neste trabalho é o de resolver o problema para múltiplos objetos, pode-se estender o Modelo 6 resultando no Modelo 7 para múltiplos objetos.

$$\max imizar\ \ z = \sum_{i=1}^{m}\sum_{j=1}^{n} l_j x_{ij} \qquad (7.1)$$

$$sujeito\ a\quad : \sum_{i=1}^{m} x_{ij} \leq 1,\ j = 1,...,n, \qquad (7.2)$$

$$\sum_{j=1}^{n} l_j x_{ij} \leq L, i = 1,...,m, \qquad (7.3)$$

$$x_{ij} = 0\ ou\ 1, i = 1,...,m.\ j = 1,...,n. \qquad (7.4)$$

$$l_j > 0, l_j \leq L, L > 0 \qquad (7.5)$$

Modelo 7. Problema do corte 0 ou 1, para múltiplos objetos em estoque.

Todos os modelos mostrados acima são adaptados para o PCU a partir de originais do problema chamado *Knapsack Problems* (Problema da Mochila). O problema da mochila

aparece em 1957 na literatura com o artigo famoso "*Discrete variable extremum problems*", de George Bernard Dantzig (DANTZIG, 1957).

Segundo Yanasse (1997), este problema surge com uma situação hipotética, na qual um muambeiro deseja carregar sua sacola (ou mochila) com itens, cujos valores de compra são ***R$ l_i, i = 1,...,m***. O valor total da compra não pode ultrapassar ***R$ L*** (por razões alfandegárias). O lucro sobre cada item é conhecido e dado por ***v_i, i = l,...m***. O muambeiro deseja maximizar seu lucro total.

Outra situação é supor que você vá acampar e tenha que escolher um conjunto de itens a serem levados, tais como: martelo, canivete, gás, roupas, corda, lanterna etc. Os itens escolhidos serão levados em uma mochila onde o fator limitante é o peso máximo que você está disposto a carregar, por exemplo, 30 Kg. Quais itens devem ser escolhidos?

4.3 Considerações

Os estudos sobre o problema do corte foram iniciados em Zonta (2000.1; 2005) necessários para realização do projeto de paredes de divisórias. Este capítulo apresentou um estudo bem mais detalhado do que já tinha sido realizado, já que possibilitou o apontamento da nomenclatura e tipologia de qual corte será resolvido, facilitando, ainda mais, o entendimento do problema.

Com todas as técnicas estudas é possível apresentar o método e o desenvolvimento necessário para chegar a uma conclusão de como será o seu funcionamento.

5 DESENVOLVIMENTO E DISCUSSÃO

Neste capítulo, todo o trabalho realizado para a apresentação do método proposto - criação de projetos de forros - será mostrado. Como já citado, o trabalho foi dividido em partes, sendo que, para a realização do corte, a realização dos desenhos e o seu mapeamento são necessários para a criação de uma estrutura de dados para armazenamento e recuperação. Com isso, será possível determinar o orçamento do projeto.

5.1 Implementação de uma Interface para realização dos Desenhos

Para a realização dos desenhos, foi testado um componente chamado TCAD XP. Este componente já é utilizado por uma comunidade de desenvolvedores de aplicativos CAD, o que garante sua continuidade e melhoria. Para testá-lo, foram criados protótipos com as funcionalidades necessárias para a implementação de um GUI (*Graphical User Inteface*) de fácil utilização. Nesta seção, será mostrada uma possível forma de como o usuário, através de um GUI, construirá um projeto para forros.

Imagina-se que o usuário dispõe de 3 formas básicas para criar uma planta baixa relacionada a um projeto de forros; a saber:

retângulos;

polígonos quaisquer;

polilinhas ortogonais;

Na Figura 40 é mostrada uma possível barra de ferramentas contendo os botões para estes 3 recursos.

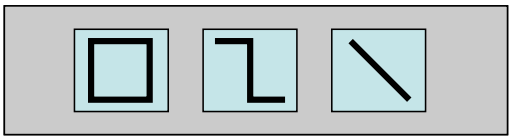

Figura 40: Possível barra de ferramentas para a criação de formas geométricas. Botões: retângulo, polilinhas ortogonais (retas adjacentes formando ângulos retos) e polilinhas genéricas (retas adjacentes com qualquer inclinação).

Com o botão retângulo, o usuário pode criar, na tela do computador, qualquer tipo de forma retangular usando o mouse. Através do botão polilinhas é possível criar uma sequência de retas adjacentes com qualquer inclinação. Pelo botão polilinhas ortogonais, também é possível criar um conjunto de retas adjacentes, porém, estas retas são sempre ou horizontais ou verticais. A Figura 41 exibe exemplos de possíveis formas que podem ser desenhadas a partir de qualquer um destes recursos.

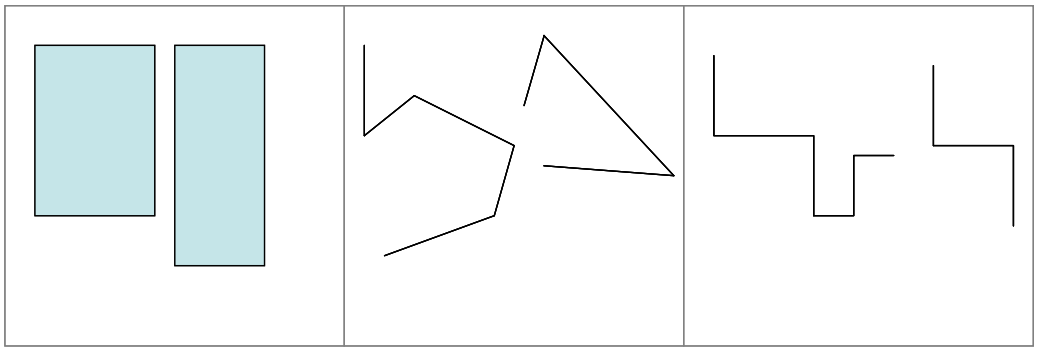

Figura 41: Possíveis formas geométricas construídas a partir dos botões mostrados na Figura 40.

O componente TCAD oferece as funcionalidades para criação destes desenhos e disponibiliza uma classe chamada TMyCAD, que contém as informações do desenho realizado, como, por exemplo, obter os pontos no espaço vetorial referente ao ponto 0 que pode ser configurado como o canto superior ou inferior esquerdo. Além das informações, o componente fornece todo o ambiente com funcionalidades que facilitam a sua utilização como, por exemplo, agrupamento de duas formas geométricas, entre outras.

A interface implementada para realização dos testes pode ser vista na Figura 42, esta interface disponibiliza a barra de ferramentas da Figura 40 e algumas funcionalidades a mais comuns em aplicativos de desenho, como por exemplo, seleção, rotação e régua, além da barra de ferramentas foram disponibilizadas algumas funcionalidades do TCAD, como por exemplo, alinhamento e zoom que possibilita ao usuário a visualização do desenho caso este

seja maior que a área disponível, este processo de zoom é tratado automaticamente pelo TCAD sem a perda das coordenadas e tamanho da obra:

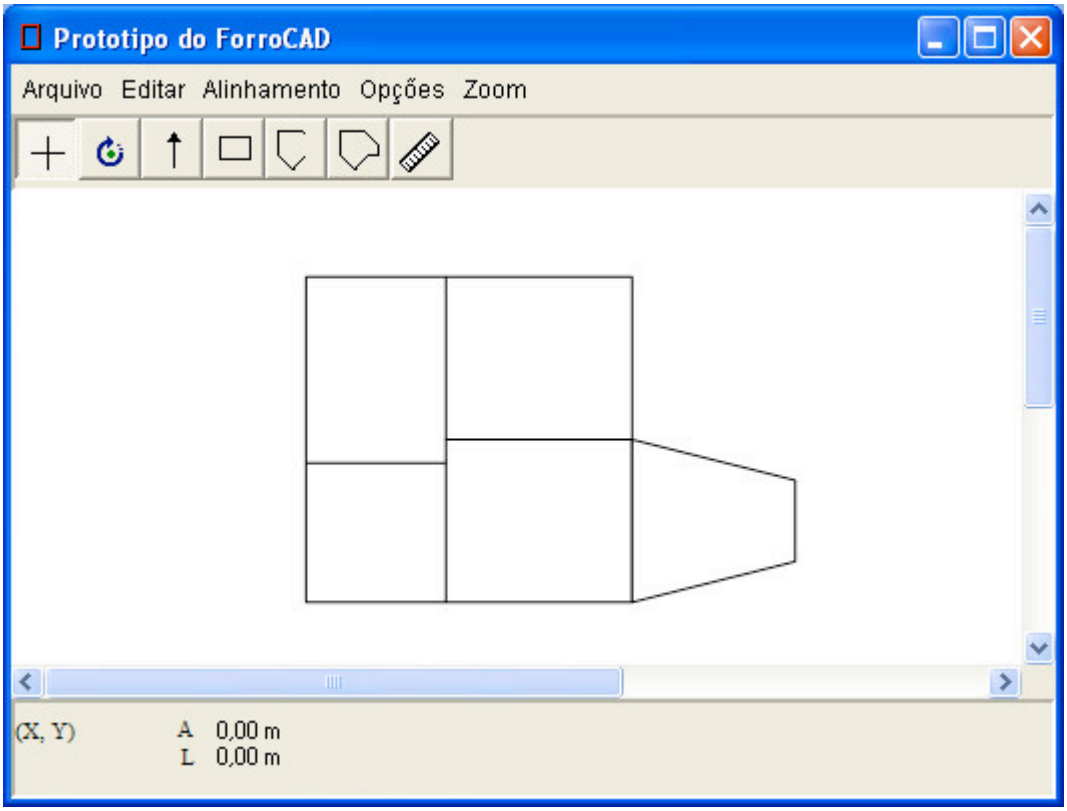

Figura 42: Protótipo desenvolvido para realização dos testes.

5.2 Representação do Projeto de Forro

A fim de se obter uma representação e mapeamento do projeto no formato que o computador possa armazenar, ou seja, criação de uma estrutura de dados e um conjunto de funcionalidades para manipulá-la.

Um projeto de forro é mostrado na Figura 43. Nesta figura foram adicionados, em sequência, um retângulo (pontos 1, 2, 3 e 4) e as polilinhas ortogonais 5, 6, 7 e 8; 9 e 10; 11 e 12. Cada número representa um ponto e cada ponto deverá ser representado internamente na forma de coordenadas ***(x, y)***.

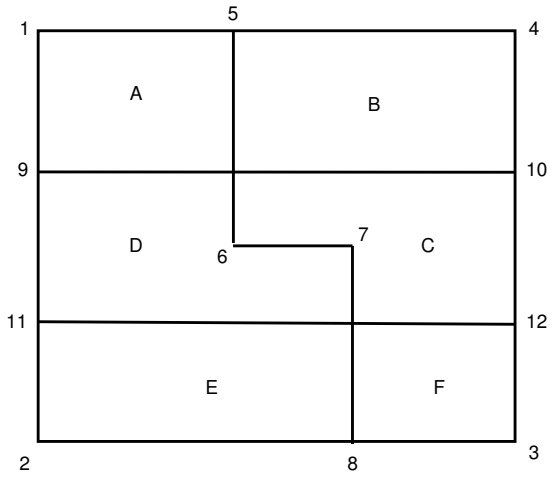

Figura 43: Planta baixa representando um projeto de forro que pode ser construído usando-se os botões retângulo e polilinha ortogonal.

Uma possível representação interna para o projeto da Figura 43 é uma estrutura de dados que contenha cada uma das quatro formas geométricas criadas:

ProjetoForro
retangulo: {1, 2, 3, 4, 1}
linha1: {5, 6, 7, 8}
linha2: {9, 10}
linha3: {11, 12}
FimProjeto;

Esta deve ser uma estrutura de dados dinâmica.

Não é difícil verificar que esta representação não é adequada. O usuário, ao criar o desenho mostrado nesta figura, delimitou cada peça do forro (cômodo), nas quais deseja calcular os pedaços e a quantidade de materiais a serem utilizados. Estas peças, que serão chamadas ambientes, foram nomeadas pelas letras A, B, C, D, E e F na Figura 43. Pela representação mostrada acima, vê-se que estas peças não estão devidamente separadas. Isto é, não estão explícitos os pontos, ou segmentos, que formam os ambientes deste projeto. O processo de identificar cada polígono de um projeto será o principal resultado deste capítulo.

Outra abordagem para a representação do projeto é desconsiderar como o usuário criou o desenho do projeto. Isto é, não importa se foi usado um retângulo ou uma das

polilinhas para a construção da planta, deve-se armazenar apenas os pontos e os segmentos que os conectam. Sendo assim desconsiderando as coordenadas de cada ponto, o projeto será visto como um grafo e poderá ser representado usando uma das formas de representação de grafos, como por exemplo, a matriz de adjacência (SWARTZFITTER, 1988). Portanto, o projeto da Figura 43 será representado pela seguinte matriz:

	1	2	3	4	5	6	7	8	9	10	11	12
1		1										
2			1									
3				1								
4	1											
5						1						
6							1					
7								1				
8												
9										1		
10												
11												1
12												

Figura 44: Representação do projeto da Figura 43.

Na matriz da Figura 44, se $a_{ij} = 1$, significa que existe um segmento de reta entre os pontos i e j. Caso $a_{ij} = 0$ (branco na matriz), não há segmento de reta entre i e j.

Toda vez que o usuário adicionar um retângulo, é necessário adicionar quatro segmentos de retas (arestas) na matriz de adjacência e um para cada reta de uma polilinha. As coordenadas de cada ponto devem ser armazenadas separadamente em um vetor.

Esta matriz armazena apenas as arestas do grafo (relação entre os vértices), no entanto, o problema de identificar os ambientes ainda não foi resolvido. Nas próximas seções será delineado um método que, a partir da matriz de adjacência, identifica cada ambiente de um projeto de forro. Tendo em vista que o forro é montado separadamente em cada ambiente, a identificação destes será de crucial importância para a determinação da quantidade de materiais e, consequentemente, do orçamento do projeto.

5.3 Interseção das Retas

Como pode ser observado na matriz de adjacência, há apenas 12 pontos representando o projeto (com nove arestas) da Figura 43. Mas existem outros pontos que devem ser identificados corretamente, aqueles onde as retas se cruzam. Sendo assim, todos os pontos de interseção devem ser determinados. Com isso, o grafo original transforma-se em um grafo planar (SWARTZFITTER, 1988). Existem oito pontos de interseção no exemplo da Figura 43. O novo projeto é mostrado na Figura 45.

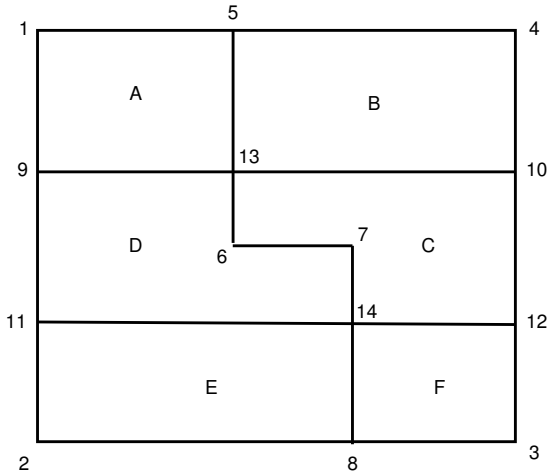

Figura 45: Planta baixa com todos os pontos de interseção entre os segmentos de retas. Os pontos 5, 9, 10, 11, 12 e 8 só eram usada por uma das retas. Agora, a partir deles, partem 3 segmentos. Dos 2 novos pontos: 13 e 14 partem 4 novos segmentos de retas.

O problema de encontrar todos os pontos de interseção de um conjunto de retas é um clássico em geometria computacional (O'ROURKE, 1994). A nova matriz de adjacência, com os dois novos pontos, é:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1					1				1					
2								1			1			
3								1				1		
4					1					1				
5													1	
6							1						1	
7														1
8														1
9											1		1	
10												1	1	
11														1
12														1
13														
14														

Figura 46: Representação do projeto da Figura 45.

Observe que a matriz da Figura 46 é triangular superior. Na verdade ela é simétrica, pois cada aresta pode ser representada tanto por (i, j) quanto por (j, i). Há, nesta matriz de adjacência, 19 arestas (lado de cada ambiente).

5.4 Identificação dos Ambientes

Por uma simples inspeção na matriz de adjacência, é possível observar que o projeto é composto por um conjunto de retas representadas na matriz. Esta representação não traz nenhuma informação de quais são os ambientes do projeto, isto é, quais são os pontos adjacentes que formam o polígono do ambiente.

Nesta seção será apresentado um algoritmo para, dada a matriz de adjacência que representa o projeto como um todo, identificar cada possível polígono que está implícito nesta matriz.

O algoritmo é, basicamente, composto de dois passos:

1. Ordenar todos os vértices adjacentes ao vértice v_i no sentido horário, com $i = 1,2,\ldots,n$, no qual n é o número de vértices;
2. Identificar os polígonos que compõem o projeto de um forro.

5.4.1 Ordenação dos Vértices

Cada vértice v_i do projeto possui p vértices adjacentes, $v_1, v_2, v_3, \ldots, v_p$, como mostra a Figura 47.

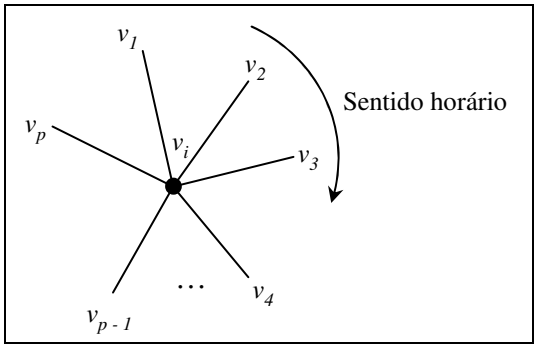

Figura 47: Vértice v_i com os seus *p* vértices adjacentes.

Para ordenar, no sentido horário, os p vértices adjacentes do vértice v_i, usa-se o produto cruzado. Através do produto cruzado é fácil verificar se o vértice (v_i, v_j) está à esquerda ou à direita do vértice (v_i, v_k). Com esta informação não é difícil elaborar um algoritmo para colocar em ordem, no sentido horário, todos os vértices incidentes ao vértice v_i. Esta ordenação deve ser realizada para todo vértice v_i do projeto, com $i = 1,2,\ldots,n$ (números de vértices do projeto). Portanto, pode-se criar um vetor de tamanho n no qual cada posição i deste vetor contém uma lista ordenada no sentido horário dos vértices adjacentes a v_i.

Por exemplo, considerando o projeto da Figura 45, obtém-se o seguinte vetor de listas ordenadas:

Tabela 3. Vetores dos vértices ordenados no sentido horário.

Vértice		Ordem no sentido horário
v_1	→	{5,9}
v_2	→	{8, 11}
v_3	→	{8, 12}
v_4	→	{10, 5}
v_5	→	{4, 13, 1}
v_6	→	{13, 7}
v_7	→	{6, 14}
v_8	→	{14, 3, 2}
v_9	→	{1, 13, 11}
v_{10}	→	{12, 13, 4}
v_{11}	→	{9, 14, 2}
v_{12}	→	{3, 14, 10}
v_{13}	→	{6, 9, 5, 10}
v_{14}	→	{8, 11, 7, 12}

Com esta lista de ordenação para cada vértice é possível criar a matriz da Figura 48 (denotada próximo):

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1					9				5					
2								11			8			
3								12				8		
4					10					5				
5	4			13									1	
6							13						7	
7						14								6
8		14	2											3
9	13										1		11	
10				12								13	4	
11		9							14					2
12			14							3				10
13					10	9			5	6				
14							12	11			7	8		

Figura 48: Matriz próximo Figura 45.

Pela matriz próximo, define-se a seguinte propriedade: $prox(i, j) = k$ se (v_i, v_k) é a próxima aresta, no sentido horário, mais a direita da aresta (v_i, v_k). Em outras palavras, o vértice v_k é o sucessor (está à direia) do vértice v_j, considerando que ambos partem do vértice v_i. Se v_j for o último vértice da lista, o vértice mais à direita será o primeiro da lista. Considerando o exemplo anterior:

$$prox(12,14) = 10$$
$$prox(9,11) = 1$$
$$prox(13,6) = 9$$

Em termos computacionais, escrever a ordenação na forma de matriz tem um custo menor para se obter, no sentido horário, o vértice v_k mais próximo do vértice v_j com ambos partindo do vértice v_i. Observar que esta matriz, como a matriz de adjacência, é altamente esparsa.

5.5 Algoritmo Proposto

Com a modelagem da estrutura de dados montada, chega o momento de o planejamento do algoritmo ser criado. A idéia, então, seria a criação de um algoritmo a partir dos passos descritos abaixo:

1. Entrada das informações através do desenho de um mapa planar de pontos e segmentos;
2. Mapeamento dos pontos e dos segmentos de reta;

3. Ordenação da lista dos pontos para obtenção dos polígonos. A ordenação seria feita a partir de cada ponto, utilizando produto cruzado;
4. Obtenção dos polígonos simples;
5. Através dos polígonos, criar uma lista individual dos pontos e segmentos que fazem parte de cada um;
6. Com cada polígono separado será possível calcular sua área e realizar o preenchimento conforme o padrão da matéria-prima que será utilizada em cada cômodo (polígono). Calcular, também, a área total do polígono principal;
7. Aplicar o problema do corte unidimensional;
8. Utilizar a quantidade de matéria prima juntamente com as armações necessárias calculadas conforme a área de cada polígono para apresentar o orçamento. Caso toda a obra seja da mesma matéria prima, as armações podem ser calculadas de forma geral.

O algoritmo acima não foi implementado da forma como descrito, já que há implementações eficientes dos algoritmos e de estruturas de dados de geometria computacional, como a CGAL. Uma tal estrutura de dados que implementa idéias descritas acima é a *Halfedge*, descrita anteriormente.

5.6 Solução utilizando a CGAL

Com a definição do algoritmo, foi possível realizar uma pesquisa envolvendo grafos planares e geometria computacional a fim de encontrar a teoria necessária para realização da identificação dos cômodos da planta baixa. Com isso, foi encontrado a CGAL que, exigiu um conhecimento mais avançado em C++ para a criação do algoritmo.

Para demonstrar a solução utilizando CGAL será utilizada a planta baixa da Figura 49. O desenho já está exibido com seus pontos cartesianos e dentro de um espaço vetorial com o ponto origem(0,0) no canto inferior esquerdo. Esta determinação dos pontos e segmentos já é realizada pelo componente TCAD através de sua classe TMyCAD. O próximo passo, então, é passar estas informações para a CGAL, já que o TCAD não apresenta a relação entre os pontos e segmentos, ou seja, no momento da ligação destes, que forma será apresentada, já

que um conjunto de segmentos podem criar um polígono. A comunicação entre o desenho realizado e a CGAL será feita através de arquivos textos delimitados que deverão ser interpretados tanto na passagem das informações, como no retorno do mapeamento realizado pelo CGAL.

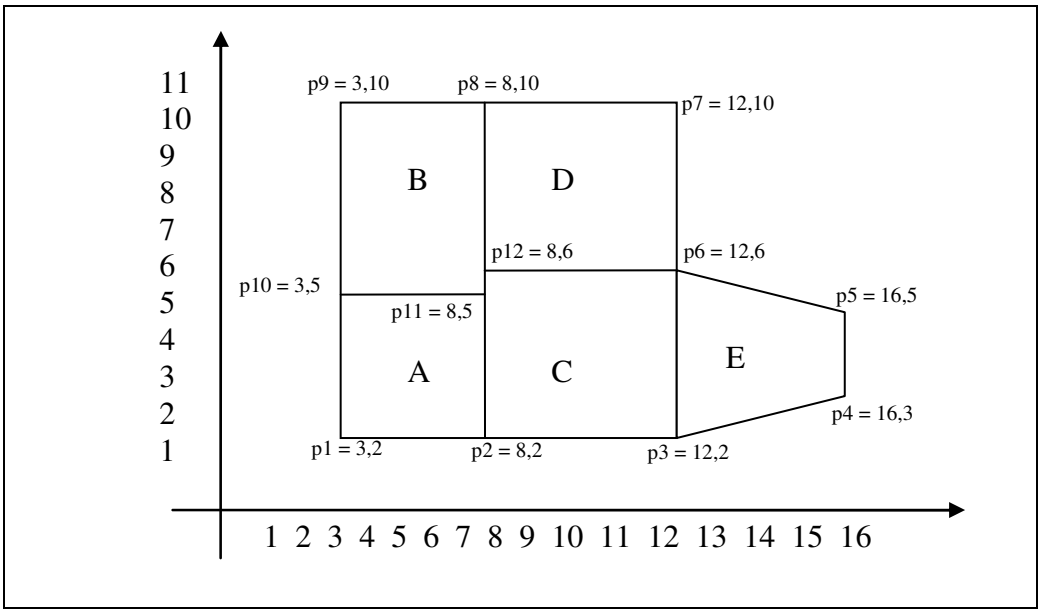

Figura 49: Planta baixa com 5 cômodos, A, B, C, D e E.

A Figura 49 demonstra uma planta baixa formada por 12 pontos ou vértices.

3,2 | 8,2 | 12,2 | 16,3 | 16,5 | 12,6 | 12,10 | 8,10 | 3,10 | 3,5 | 8,5 | 8, 6.

Estes vértices separados pelo TCAD como pontos, formam a estrutura de dados principal do problema, o formato acima será enviado para o CGAL como um texto que será interpretado ponto a ponto através do separador **|** (pipe).

Outra informação gerada pela Figura 49 são os segmentos que são identificados a seguir:

retangulo: {p1, p3, p7, p9}
poligono: {p3, p4, p5, p6}
linha1: {p2, p8}
linha2: {p10, p11}
linha3: {p12, p6}

A idéia é, através dos vértices, iniciando por 3,2, por exemplo, identificar uma lista de vértices que formem um polígono. Neste caso, uma nova estrutura seria formada com cada polígono separado.

Exemplo:

Vértices: p1 = 3,2 | p10 = 3,5 | p11 = 8,5 | p2 = 8,2 polígono A.

Neste caso, esta planta baixa seria formada de 5 polígonos:

A: p1 | p10 | p11 | p2;

B: p10 | p9 | p8 | p12 | p11;

C: p3 | p2 | p11 | p12 | p6;

D: p8 | p7 | p6 | p12;

E: p6 | p5 | p4 | p3.

No fim, com estas 5 estruturas, será possível determinar a área individual de cada polígono e a área total do projeto do forro.

Neste exemplo, não foi utilizada interseção entre polígonos, mas isso pode ocorrer, sendo que deverão ser identificadas as ilhas (cômodos) e os buracos (polígono principal) para isolar os vértices de cada polígono.

Os métodos aqui descritos foram baseados no trabalho de Hanniel (2000) que trata da utilização do CGAL para mapas planares e aponta dois objetos básicos para isso:

Kernel::Point_2	**Point_2;**
Traits_2::X_monotone_curve_2	**Segment_2.**

Definem-se os 12 pontos referentes à planta baixa utilizando as informações do texto enviado para o CGAL, veja abaixo como ficarão os pontos do exemplo proposto:

Point_2 p1(3,2),p2(8,2),p3(12,2),p4(16,3),p5(16,5),p6(12,6),
p7(12,10),p8(8,10),p9(3,10),p10(3,5),p11(8,5),p12(8,6);

Define-se um vetor com os segmentos gerados pelas formas geométricas, no caso do retângulo, por exemplo, serão geradas 4 arestas (p1, p3), (p3, p7), (p7, p9) , (p9, p1), veja o exemplo do vetor de segmentos gerados para planta baixa proposta:

Segment_2 segments[] = {Segment_2 (p1, p3),
Segment_2 (p1, p9),
Segment_2 (p2, p8),
Segment_2 (p3, p7),
Segment_2 (p10, p11),
Segment_2 (p9, p7),
Segment_2 (p12, p6),
Segment_2 (p3, p4),
Segment_2 (p4, p5),
Segment_2 (p5, p6)};

Após este mapeamento foi feita a inserção dos segmentos na estrutura de arranjo **CGAL::Arrangement_2<Traits_2> Arrangement_2.** Desta forma, CGAL tem todas as informações necessárias para processar o desenho.

A CGAL possui vários predicados que podem auxiliar na construção de uma solução, um deles pode ser a utilização dos métodos de pré condição como, por exemplo, verificar se o arranjo foi mapeado de forma correta. **CGAL_precondition (arr.is_valid());**

Várias informações podem ser retiradas do arranjo. O código da Figura 50 exibe como obter as informações dos vértices.

```
1. typename Arrangement::Vertex_const_iterator vit;
2. cout << arr.number_of_vertices() << " vertices:" << endl;
3. for (vit = arr.vertices_begin(); vit != arr.vertices_end(); ++vit)
   {
4.   cout << "(" << vit->point() << ")";
5.   if (vit->is_isolated())
6.     cout << " - Isolado." << endl;
     else
7.       cout << " - grau " << vit->degree() << endl;
   }
```

Figura 50: Código fonte para obter as informações dos vértices.

Para entender o funcionamento da CGAL, o código será facilmente explicado:

1. É criado um *iterator* para percorrer os vértices;

2. É utilizado o método *number_of_vertices* para exibir a quantidade de vértices do arranjo;

3. O comando ***for*** percorre os vértices do inicio até o fim;

4. São exibidos os vértices do desenho;

5, 6. Verifica se o vértice está isolado, sendo que, neste exemplo, não temos nenhum, e para este trabalho isso não vai ocorrer.

7. Mostra o grau de vizinhança do vértice, ou seja, mostra a quantidade de vértices adjacentes que determinado vértice possui.

O resultado obtido com o código fonte da Figura 50 pode ser visto abaixo. São 12 vértices, sendo que o vértice **3,5** é de **grau 3** por apresentar adjacência com os vértices **3,2 | 8,5 | 3,10**.

12 vértices:
p1 = (3,5) - grau 3
p10 = (3,2) - grau 2
p9 = (3,10) - grau 2
p2 = (8,2) - grau 3
p11 = (8,5) - grau 3
p12 = (8,6) - grau 3
p8 = (8,10) - grau 3
p3 = (12,2) - grau 3
p6 = (12,6) - grau 4
p7 = (12,10) - grau 2
p4 = (16,3) - grau 2
p5 = (16,5) - grau 2

A vizinhança ou adjacência de cada vértice pode ser exibida com:

```
1. void print_neighboring_vertices (typename Arrangement::Vertex_const_handle v)
{
2. if (v->is_isolated()){
     cout << "O vertice (" << v->point() << ") e isolado" << endl;
     return;
   }
3. typename Arrangement::Halfedge_around_vertex_const_circulator
first,curr;
4. typename Arrangement::Vertex_const_handle u;

5. cout << "Os vizinhos do vertice (" << v->point() << ") sao:";
6. first = curr = v->incident_halfedges();
7. do{
     u = curr->source();
     cout << " (" << u->point() << ")";
     ++curr;
  } while (curr != first);
  cout << endl;

  return;
}
```

Figura 51: Código fonte para obter as informações da vizinhança de um vértice.

O código da Figura 51 tem a seguinte explicação:

1. Declaração do cabeçalho da função que é chamada para cada vértice, passado por parâmetro;
2. Verifica se o vértice é isolado, se sim, nem continua, porque um vértice isolado não possui vizinhos;
3. É a primeira utilização da *Halfedge*. Com ela será possível criar um *iterator* para circular pelo vértice;
4, 5, 6, 7. Obtém a estrutura do vértice para exibir sua vizinhança.

Com o resultado da vizinhança é fácil identificar o grau de cada vértice, ou seja, é necessário contar a vizinhança do vértice para se obter seu grau.

Os vizinhos do vértice (p10 = 3,5) são: (p1 = 3,2) (p9 = 3,10) (p11 = 8,5)
Os vizinhos do vértice (p1 = 3,2) são: (p10 = 3,5) (p2 = 8,2)
Os vizinhos do vértice (p9 = 3,10) são: (p10 = 3,5) (p8 = 8,10)
Os vizinhos do vértice (p2 = 8,2) são: (p1 = 3,2) (p11 = 8,5) (p3 = 12,2)
Os vizinhos do vértice (p11 = 8,5) são: (p2 = 8,2) (p10 = 3,5) (p12 = 8,6)
Os vizinhos do vértice (p12 = 8,6) são: (p11 = 8,5) (p8 = 8,10) (p6 = 12,6)
Os vizinhos do vértice (p8 = 8,10) são: (p12 = 8,6) (p9 = 3,10) (p7 = 12,10)
Os vizinhos do vértice (p3 = 12,2) são: (p2 = 8,2) (p6 = 12,6) (p4 = 16,3)
Os vizinhos do vértice (p6 = 12,6) são: (p3 = 12,2) (p12 = 8,6) (p7 = 12,10) (p5 = 16,5)
Os vizinhos do vértice (p7 = 12,10) são: (p6 = 12,6) (p8 = 8,10)
Os vizinhos do vértice (p4 = 16,3) são: (p3 = 12,2) (p5 = 16,5)
Os vizinhos do vértice (p5 = 16,5) são: (p4 = 16,3) (p6 = 12,6)

Outra informação importante que pode ser obtida com o arranjo criado é a quantidade de arestas e quais vértices responsáveis por sua formação. O código da Figura 52 demonstra como isso pode ser obtido através da CGAL:

```
1.  typename Arrangement::Edge_const_iterator eit;
2.  cout << "\n\n" << arr.number_of_edges() << " arestas:" << endl;
3.  for (eit = arr.edges_begin(); eit != arr.edges_end(); ++eit)
      cout << "[" << eit->curve() << "]" << endl;
```

Figura 52: Código fonte para obter as informações das arestas.

O código da Figura 52 tem a seguinte explicação:

1. É criado um *iterator* para percorrer as arestas;
2. É utilizado o método *number_of_edges* para exibir a quantidade de arestas do arranjo;
3. O comando ***for*** percorre as arestas do inicio até o fim exibindo suas informações;

O resultado obtido com o código é a informação da quantidade de arestas e quais vértices formam cada uma delas.

16 arestas:
[p1 = 3,2 p10 = 3,5]
[p10 = 3,5 p9 = 3,10]
[p1 = 3,2 p2 = 8,2]
[p2 = 8,2 p11 = 8,5]
[p10 = 3,5 p11 = 8,5]
[p11 = 8,5 p12 = 8,6]
[p12 = 8,6 p8 = 8,10]
[p9 = 3,10 p8 = 8,10]
[p2 = 8,2 p3 = 12,2]
[p3 = 12,2 p6 = 12,6]
[p12 = 8,6 p6 = 12,6]
[p6 = 12,6 p7 = 12,10]
[p8 = 8,10 p7 = 12,10]
[p3 = 12,2 p4 = 16,3]
[p4 = 16,3 p5 = 16,5]
[p5 = 16,5 p6 = 12,6]

As informações acima foram exibidas somente para demonstração da facilidade oferecida pela CGAL e, basicamente, justificar o porquê de sua escolha. A parte mais importante do código criado será vista a seguir. Este código é responsável pela determinação das informações do polígono principal, chamado de buraco, e de cada polígono individual.

```
1.  typename Arrangement::Face_const_iterator    fit;
2.  cout << "\n\n" << arr.number_of_faces() << " faces:" << endl;
3.  for (fit = arr.faces_begin(); fit != arr.faces_end(); ++fit)
      print_face<Arrangement> (fit);
```

Figura 53: Código fonte para obter as informações de cada face.

O código da Figura 53 tem a seguinte explicação:

1. É criado um *iterator* para percorrer as faces de cada vértice;
2. É utilizado o método *number_of_faces* para exibir a quantidade de faces do arranjo;
3. O comando ***for*** percorre as faces chamando a função que exibirá as informações contidas na face passada por parâmetro;

O código da Figura 54 mostra a função que exibe as faces e a função auxiliar que realiza esta tarefa.

```
1. void print_ccb (typename Arrangement::Ccb_halfedge_const_circulator circ)
{
  typename Arrangement::Ccb_halfedge_const_circulator curr = circ;
  typename Arrangement::Halfedge_const_handle he;

  cout << "(" << curr->source()->point() << ")";
  do
  {
    he = curr;
    cout << " [" << he->curve() << "]\n(" << he->target()->point() <<      ")";

    ++curr;
  } while (curr != circ);
  cout << endl;
  return;
}



2. void print_face (typename Arrangement::Face_const_handle f)
{
3. if (f->is_unbounded())
     cout << "Face sem limite. " << endl;
   else
  {
     cout << "Borda externa:\n";
4.   print_ccb<Arrangement> (f->outer_ccb());
  }

5. typename Arrangement::Hole_const_iterator hole;
   int index = 1;

6. for (hole=f->holes_begin();hole!=f->holes_end();++hole,++index)
   {
     cout << "Buraco #\n" << index << ": ";
     print_ccb<Arrangement> (*hole);
   }

7. typename Arrangement::Isolated_vertex_const_iterator  iv;

8. for (iv = f->isolated_vertices_begin(), index = 1; iv != f->isolated_vertices_end();
++iv, ++index)
    cout << "Vertice isolado #\n" << index << ": " << "(" << iv->point() << ")" <<
endl;
  return;
}
```

Figura 54: Código fonte da função que exibe as informações de uma face passada por parâmetro.

O código da Figura 54 tem a seguinte explicação:

1. Cabeçalho da função que recebe uma face como parâmetro e exibe os vértices que formam a aresta com determinado vértice, utilizando CCB;

2. Cabeçalho da função que utiliza a função do ponto 1;

3. Utiliza o método para verificar se a face oferece um polígono sem limite, no caso, o polígono principal;

4. Exibe a divisa externa da face chamando a função do ponto 1;

5. Cria um *iterator* para verificar os vértices do polígono principal;

6. O comando ***for*** percorre os polígonos e determina quais são principais, no caso do exemplo em questão será identificado somente um buraco;

7. Cria um *iterator* para exibir os polígonos que são responsáveis pela construção do polígono principal que, no caso, são chamados de polígonos isolados;

8. O comando ***for*** percorre os vértices com divisa externa para exibir as informações de cada um, por exemplo, o ponto 3,2 mesmo fazendo parte do polígono principal, também possui divisa externa.

Os códigos acima são os mais úteis do ponto de vista deste trabalho. Caso a CGAL não tivesse sido utilizada o código dificilmente teria o tamanho e o desempenho que foi obtido. Abaixo será visto as informações geradas por este resultado, facilitando sua visualização dentro de cada figura. Será exibido o polígono principal e posteriormente cada polígono separado.

Polígono principal:
6 faces.
Face sem limite. Buraco # 1:

Vértices	Arestas
(3, 2)	[3,2 3,5]
(3, 5)	[3,5 3,10]
(3,10)	[3,10 8,10]
(8,10)	[8,10 12,10]
(12,10)	[12,6 12,10]
(12,6)	[16,5 12,6]
(16,5)	[16,3 16,5]
(16,3)	[12,2 16,3]
(12,2)	[8,2 12,2]
(8,2)	[3,2 8,2]
(3,2)	

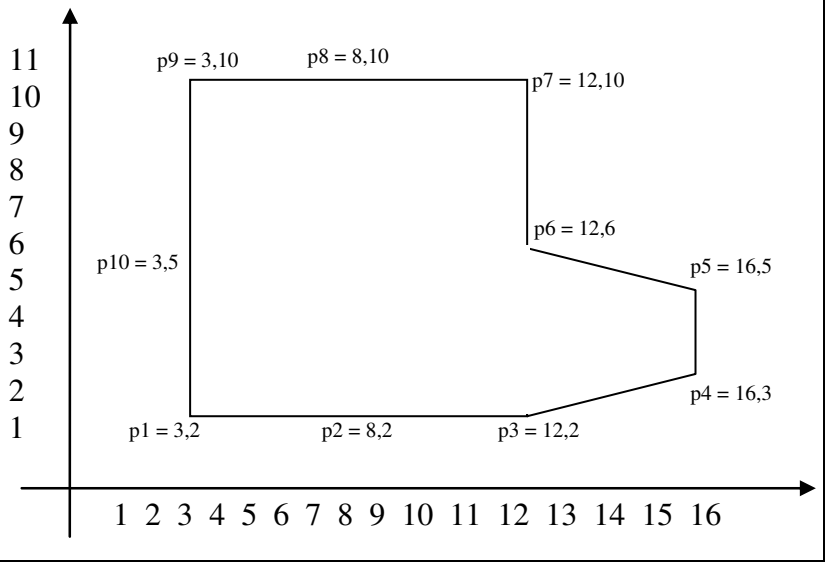

Figura 55: Informações do polígono principal, seus (vértices) e suas [arestas].

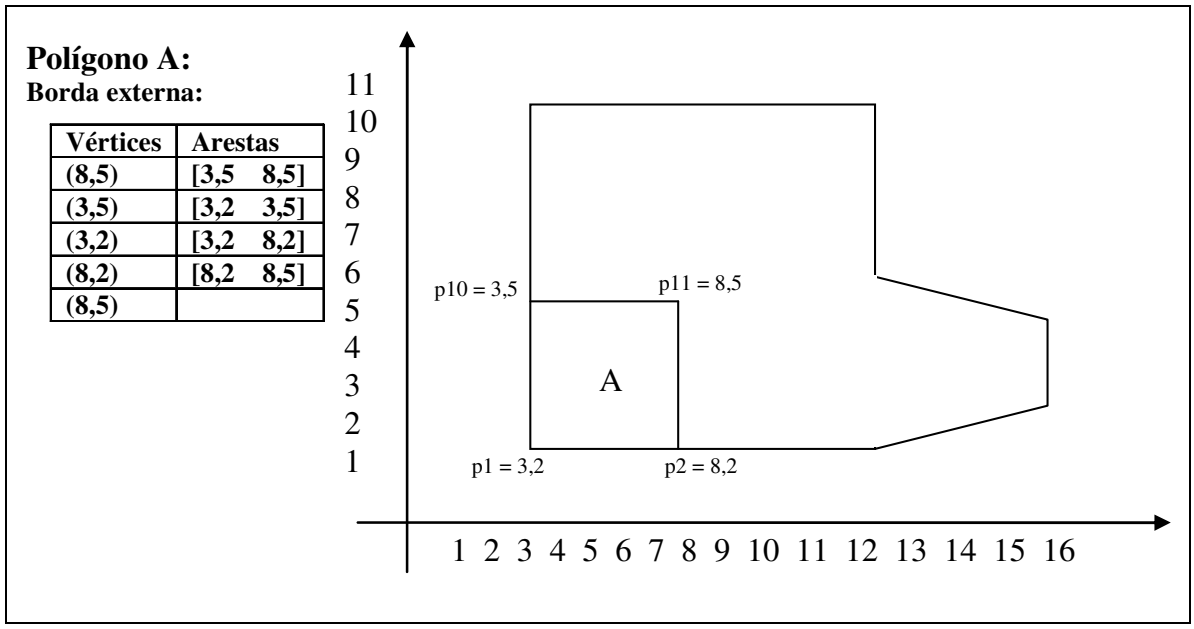

Figura 56: Informações do polígono A, seus (vértices) e suas [arestas].

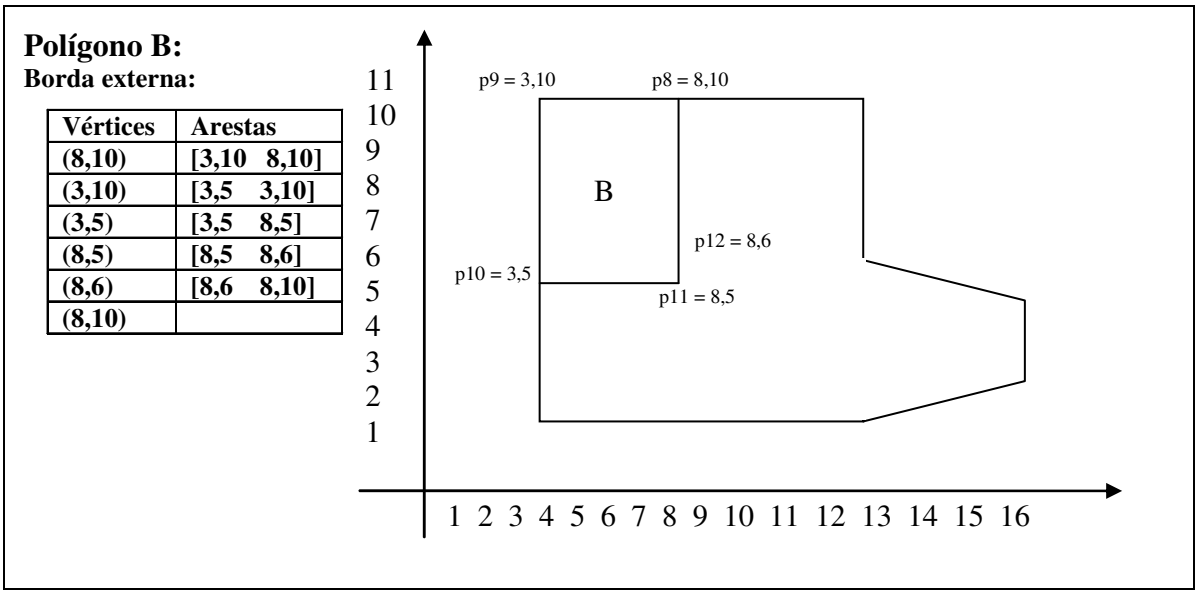

Figura 57: Informações do polígono B, seus (vértices) e suas [arestas].

Polígono C:
Borda externa:

Vértices	Arestas
(12,6)	[8,6 12,6]
(8,6)	[8,5 8,6]
(8,5)	[8,2 8,5]
(8,2)	[8,2 12,2]
(12,2)	[12,2 12,6]
(12,6)	

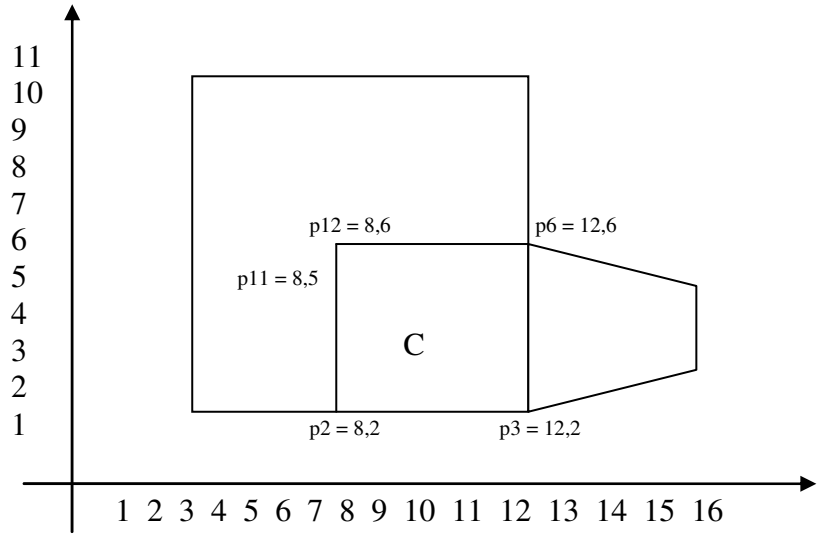

Figura 58: Informações do polígono C, seus (vértices) e suas [arestas].

Polígono D:
Borda externa:

Vértices	Arestas
(12,10)	[8,10 12,10]
(8,10)	[8,6 8,10]
(8,6)	[8,6 12,6]
(12,6)	[12,6 12,10]
(12,10)	

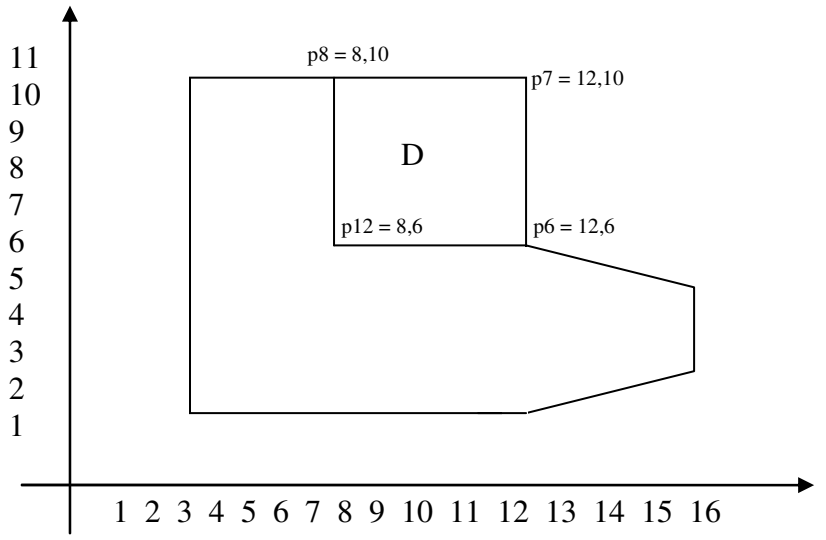

Figura 59: Informações do polígono D, seus (vértices) e suas (arestas).

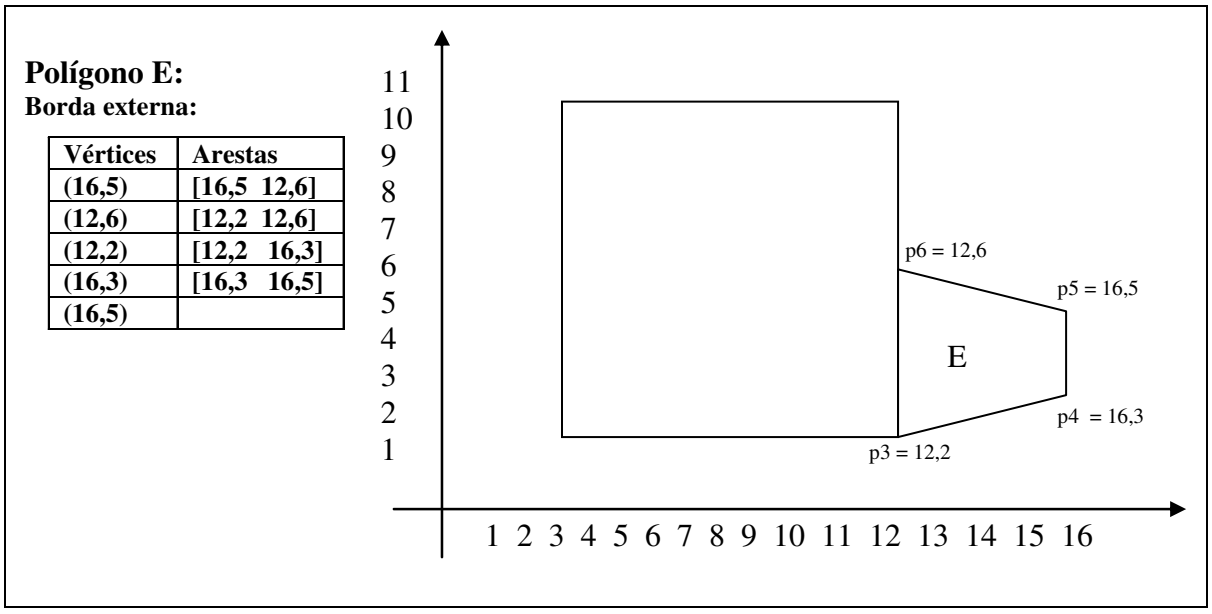

Figura 60: Informações do polígono E, seus (vértices) e suas [arestas].

5.7 Preenchimento

Com o polígono principal e os cômodos localizados, o preenchimento pode ser realizado para um cômodo específico ou para a obra inteira.

O algoritmo do preenchimento faz, basicamente, o que o próprio nome diz, preenche a área determinada com o material desejado para o forro (perfis e placas do forro). O preenchimento deverá obedecer o tamanho padrão de uma peça de forro, ou seja, ela pode tanto ser utilizada de forma inteira como também em pedaços, tais pedaços serão determinados através da solução de um PC. O algoritmo segue a teoria dos algoritmos gulosos, na qual conforme for encontrando espaço suficiente para encaixar o material fará o encaixe.

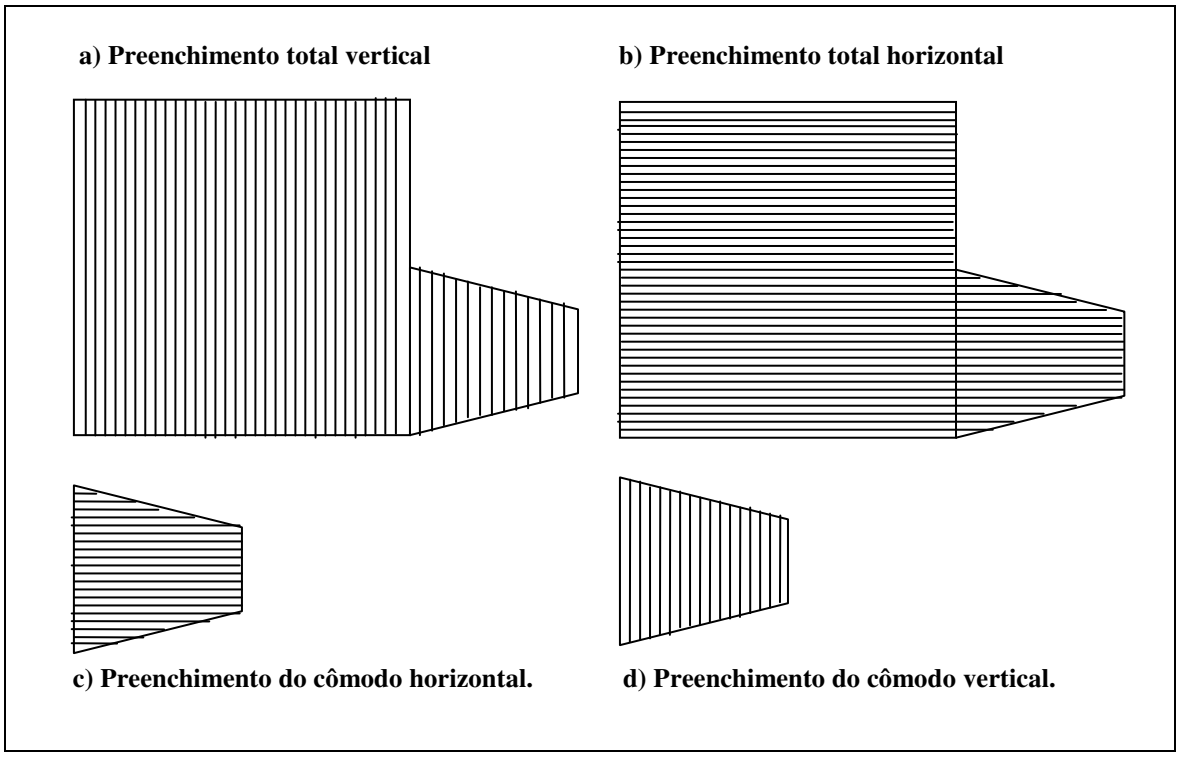

Figura 61: Formas de preenchimentos.

Várias formas de preenchimento serão permitidas, como pode ser visto através da Figura 61. Aqui é possível visualizar o preenchimento total da obra tanto na horizontal como na vertical. Isso pode dar diferença no momento da distribuição do material e modificar o valor do orçamento. Outro ponto importante é que os cômodos podem receber um preenchimento individual, o que permite que a obra seja realizada com diversas formas e tipos de materiais, de acordo com o desejo do cliente ou conforme o aplicativo indicar menor valor.

5.8 Algoritmos utilizados na resolução do PCU

Com o desenho realizado, a identificação dos polígonos e o preenchimento dos mesmos, obtem-se os dados para a aplicação do corte. Nesta seção será feita a apresentação de cada procedimento utilizado para resolução do PCU.

Para iniciar os estudos sobre o PCU utilizando algumas heurísticas foram escolhidos 3 algoritmos para fazer a comparação entre seus resultados. Todos os algoritmos admitem que os valores são inicialmente inteiros positivos, nenhum item será maior que o limite da solução

e, por último, todos os itens devem estar em ordem decrescente. O processo que garante isso será realizado com os dados de entrada gerados pelos desenhos. A partir da entrada, toda a padronização obedecerá ao tipo de material do forro.

5.8.1 Algoritmo *First Fit Decreasing* (FFD)

O algoritmo FFD, já em sua forma original, resolve o problema do corte para múltiplos objetos em estoque, isto é, corta diversos objetos. O funcionamento deste algoritmo é baseado na idéia de que o primeiro item não utilizado e útil para a solução corrente, ou seja, se encaixa nela, será utilizado. O algoritmo FFD utiliza apenas a restrição física do problema, não fazendo mais nenhuma restrição aos itens que serão utilizados na solução.

Na Figura 62 é possível ver o pseudocódigo com o funcionamento básico do algoritmo FFD. São dois laços nos quais um controla o número de itens a serem produzidos e as novas soluções a serem criadas e o outro, mais interno, se encarrega da verificação da possível utilização dos itens que ainda não foram utilizados. A execução inicia com a criação e ordenação dos itens a serem produzidos de forma decrescente, logo após é chamado o algoritmo FFD. Nele será feita uma tentativa de encaixar o item corrente em alguma das soluções já existentes, caso contrário, abre-se uma nova solução.

```
início
  inteiro: i,j, numitens;
  boleano: achou,usado;
  tipo vet = vetor [] inteiro;
  vet: lista_itens, sl_corrente;
  para i de 1 até numitens faça
  início
     achou := Falso; //Flag para verificar se o pedaço se encaixou em alguma solução*/
     para j de 1 até k  faça
     início
          se lista_itens [i] <= sl_corrente[j] então
          início
             achou := True; //Foi usado quer dizer que achou uma solução que se encaixa
             sl_corrente[j] := sl_corrente[j] - lista_itens[i];//Diminui o valor da solução
             aborta;
          fim-se;
     fim-para;

     se não achou então //Se não foi achado nenhuma solução é iniciado uma nova
     início
          K := K + 1;
          AbreNovaSolucao;
     fim-se;
  fim-para;
fim.
```

Figura 62. Pseudocódigo do algoritmo FFD utilizado.

5.8.2 Algoritmo Guloso (AG)

Como o próprio nome sugere, o algoritmo é guloso porque abocanha o primeiro item viável seguindo uma ordem dada, sem se preocupar com o que vai acontecer com a solução apontada como viável. Isso quer dizer que o algoritmo apresentará somente uma solução. Porém, não se pode afirmar que ela é ótima. Com essa forma de funcionamento pode-se identificar que o consumo de tempo para resolução do PCU utilizando o algoritmo guloso é determinado por O(n), não levando em conta o tempo O(n log(n)) utilizado para ordenar os itens a serem utilizados.

O funcionamento do AG encontrado em Yanasse (1997) segue a mesma idéia do FFD. O primeiro item não utilizado e útil para solução será utilizado. Ele possui apenas uma

diferença que, em sua forma original, resolve apenas um objeto em estoque. De modo geral, aplica o corte em apenas um objeto.

Na Figura 63 pode ser visto o pseudocódigo do AG. Sua execução será iniciada após a criação e ordenação decrescente dos itens a serem produzidos. Este procedimento é igual à utilização do FFD. A diferença entre o AG e FFD é que, o AG resolve apenas um objeto por execução, a execução deverá ser realizada até o termino dos itens, que significa a produção total dos itens necessários. Neste caso, a cada execução será apresentada uma solução que não sofrerá mais nenhuma alteração durante o restante do processo.

```
início
  inteiro: i, somasolucao, utilziados, numitens, L;
  tipo vet = vetor [] inteiro;
  vet: lista_itens, lista_utilizados;
  utilizados := 0;
  enquanto utilizados >= numitens faça //Enquanto tiver itens continua procura
  início
    somasolucao := 0; //Reinicializa a soma para verificar a nova solução
    para i de 1 até numitens faça
    início
        se (somasolucao + lista_itens[i]<=L) e (lista_utilizados[i] = 0) então
        início
            somasolucao := somasolucao + lista_itens [i];
            lista_utilizados[i] := 1; //Marca item como utilizado
            utilizados := utilizados + 1;
        fim-se;
        //Verifica se é possível abortar a procura por já ter achado a solução
        se somasolucao = L então
           aborta;
    fim-para;
  fim-enquanto;
fim.
```

Figura 63. Pseudocódigo do AG utilizado.

5.8.3 Algoritmo usando Limitante de Dantzig (LD)

Visto em ("*Discrete variable extremum problems*" de George Bernard Dantzig apud YANASSE, 1997). O seu funcionamento utiliza uma estrutura de quebra de fluxo conhecida

como GOTO. O GOTO é considerado uma forma de programação não estruturada por utilizar saltos durante a execução. O LD utiliza, também, um procedimento bastante conhecido pelos programadores de PROLOG, o *backtracking*, considerado o responsável pelo aumento da complexidade do algoritmo por insistir na busca de uma solução considerada viável. Este procedimento faz a retomada de uma solução para tentar melhorá-la.

O LD é executado da seguinte forma:

Forward move: inserção da maior quantidade possível (inteira) de itens consecutivos à solução que está sendo investigada (Solução Corrente);

Backtracking move: remoção do último item inserido na solução corrente, realizando uma nova procura por um item de melhor utilidade. O *Backtracking move* é a chamada mais importante do algoritmo. É nela que se aplica o retorno à exploração do resultado;

Limitante Superior: quando um determinado item não puder ser agregado à solução corrente, calcula-se o limitante associado. Faz-se, então, uma comparação desta última com a melhor solução encontrada até então. Se não for potencialmente possível melhorar a melhor solução, faz-se um *backtracking*, caso contrário um *forward move* ocorre;

Testes de parada: são necessários dois testes de parada:

Término parcial: se o último item tiver sido considerado, testa-se a **solução corrente** com a melhor solução, para uma possível atualização;

Término geral: se não for possível fazer *backtrack* algum.

5.9 Comparações entre os algoritmos para resolução do PCU

As comparações entre os três algoritmos já foram realizadas em Zonta (2000.1; 2005). Aqui, uma das tabelas com dados obtidos nos experimentos realizados será exibida. Esta tabela ajudará na justificativa do porquê da escolha do algoritmo LD. Para os testes das soluções foram utilizados valores gerados aleatoriamente. Os testes foram realizados com um estoque de 500, 1000 e 2000 itens a serem produzidos para a criação de um objeto de tamanho *L = 150 cm*. A tabela que será exibida é o teste com 2000 itens.

Tabela 4. Tabela com os resultados para execução dos algoritmos com uma entrada pequena de 2000 itens em estoque.

Algoritmos	**FFD**	**LD**	**AG**
Itens	2000	2000	2000
Objetos	1038	1032	1038
Valor máximo (%)	100	100	100
Perda (%)	3,18	1,91	1,34
Total de aproveitamento	96,82	98,09	98,66

Segue a descrição de cada linha da tabela que representa um parâmetro para análise e comparação dos algoritmos:

Itens: Itens de tamanhos menores em estoque a serem produzidos. Aos itens em questão, por causa de um problema encontrado na prática, foi estipulado um intervalo de *L >= 1 cm* e *L <= 150 cm*;

Objetos: Objetos necessários para produzir os itens menores em estoque. Tamanho dos objetos *L = 150 cm*. Estes objetos podem ser: barras de ferro, vidros, tecidos, todos cortados em apenas uma direção, somente na horizontal ou na vertical;

Valor máximo (%): Parâmetro que avalia, dentro das soluções, qual foi o topo de utilização dos objetos. Uma solução, por exemplo, pode apresentar uma perda considerada pequena mas que deixou sobra na maioria dos objetos;

Perda (%): A perda é avaliada no montante do resultado, verificando quantos % de sobras foram obtidos nos objetos;

Total de aproveitamento (%): Com este parâmetro pode-se ter uma idéia de qual objeto a solução começou a ter sobra e avaliar junto aos outros parâmetros a eficiência e eficácia dos algoritmos.

Nos resultados apresentados na Tabela 4 pode-se visualizar que o número de objetos necessários para produzir os itens gerados são menores. Isso já seria um resultado importante para justificar a decisão pelo algoritmo LD. Esse resultado surge devido ao processo de *backtracking*, principal responsável pelas várias tentativas, possibilitando que peças de menor valor consigam ser utilizadas. Este processo também é responsável pela complexidade e lentidão deste algoritmo. Nos primeiros testes, a utilização do LD apresentou tempos de execução que poderiam inviabilizar seu uso. Com isso, um teste de parada a mais no algoritmo foi realizado, sendo que, em determinado número de iterações, o algoritmo já pode estar apresentando um resultado viável. Isso fez com que o tempo de execução do algoritmo caísse, tornando possível sua utilização. Este número de iterações pode ser configurado pelo usuário do sistema conforme avaliação do resultado.

6 CONCLUSÃO

Neste trabalho foi descrito um método para o desenvolvimento de um programa de computador para projeto e orçamento de forros modulares e não modulares. Fez-se uso de vários conceitos e métodos de áreas relacionadas, como grafos planares e geometria computacional.

Este trabalho demonstrou que uma cooperação entre empresa e academia pode gerar um bom resultado que não necessita ser apenas prático, isto é, pode oferecer uma boa referência e interessante pesquisa.

Especificamente, foi desenvolvido um algoritmo para a determinação individual de todos os cômodos que compõe um projeto de forro. Tal determinação é importante, pois, na criação do projeto, o usuário pode inserir retângulos e/ou segmentos de reta para construir, de forma fácil e rápida – através de interfaces gráficas: GUI –, o projeto do forro, sem se preocupar em inserir individualmente os dados de cada cômodo que forma o forro. Desta forma, o processo de determinação dos cômodos é automatizado.

Uma vez determinado individualmente cada cômodo (polígono) – que é internamente representado como uma lista de coordenadas cartesianas no plano – torna-se fácil calcular a quantidade de material que deve ser usada no projeto de forro. Tal quantidade é calculada individualmente para cada cômodo do projeto e soma de cada uma representa a quantidade total do material da obra. Cabe lembrar que a quantidade de alguns destes materiais – perfis principalmente – são calculados usando métodos de pesquisa operacional, isto implica que tais quantidades são determinadas para serem mínimas ou próximas da mínima.

Comparado com outros sistemas existentes em outros segmentos como confecção e têxtil (AUDACES, 2009; RZSISTEMAS, 2009), a idéia é realizar a entrada de forma visual para o usuário sem necessitar que o desenho seja feito em outro aplicativo e os itens sejam catalogados para serem digitados manualmente. Este modelo já demonstrou ser funcional para as paredes de divisórias, na qual um projeto que necessitava de 3 horas para ser realizado pôde ser apresentado em 15 minutos. Além do fator tempo, o projeto utilizado como referência apresentou uma queda de até 10% de material. Os testes aqui realizados demonstraram que isso será possível para o forro, principalmente pela possibilidade do responsável pelo orçamento apresentar ao cliente diversas formas, ou seja, tratar a obra como um todo ou dividi-la em cômodos.

O método resultante deste trabalho, então, se resume ao algoritmo abaixo:

1. Entrada das informações através do desenho de um mapa planar de pontos e segmentos, utilizando Delphi com o componente TCAD e sua classe TMyCAD;
2. Mapeamento dos pontos e dos segmentos de reta utilizando as informações do passo 1 dentro do C++ juntamente com a CGAL;
3. Através da CGAL, obter todos os polígonos (ambientes ou cômodos) do projeto do forro;
4. Através dos polígonos, criar uma lista individual dos pontos e segmentos que fazem parte de cada um, criando uma sub-divisão do polígono principal utilizando a CGAL;
5. Realizar o preenchimento conforme o padrão da matéria-prima que será utilizada em cada cômodo (polígono). Calcular, também, a área total do polígono principal;
6. Aplicar o problema do corte unidimensional utilizando o algoritmo LD, por ser uma solução que realiza diversas tentativas para o mesmo corte, apresentando resultados melhores que o AG e o FFD;
7. Utilizar a quantidade de matéria prima necessária juntamente com as armações calculadas, conforme a área de cada polígono, para apresentar o orçamento. Caso toda a obra seja da mesma matéria prima, as armações podem ser calculadas de forma geral. Este ponto ainda será realizado quando for finalizada a construção da ferramenta ForroCAD.

6.1 Trabalhos Futuros

Há varias possibilidades de futuros trabalhos a partir do descrito aqui. O principal sem dúvida é a aplicação do método e criação da ferramenta (ForroCAD) para uso na prática por alguma empresa. Outro ponto a considerar, foram os estudos da CGAL, tornando possível a aplicação desta ferramenta em vários outros tipos de aplicação.

Outro desafio que ficou para trabalhos futuros pode ser visto na Figura 64, que traz um exemplo de três polígonos e uma circunferência. Este exemplo é pouco comum nos projetos de forro. Com o crescimento do escopo deste trabalho, o desafio de identificar circunferência ficou para um segundo momento. Dois pontos para este tipo de problema já foram estudados, o desenho da circunferência e os algoritmos existentes na CGAL, que também apresentam soluções para curvas. Juntamente com as circunferências terá que ser incluído o corte de duas dimensões, devido a diversas sobras de material por causa do formato geométrico do círculo.

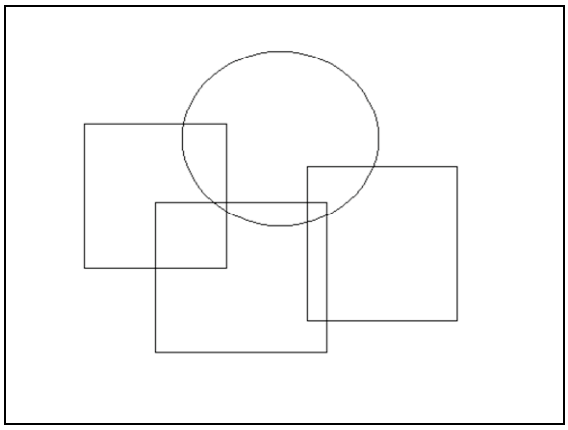

Figura 64. Planta baixa com três cômodos retangulares e um circular.

REFERÊNCIAS

ALOISE DARIO. **Página pessoal do professor Dário José Aloise**. Disponível em: http://www.dimap.ufrn.br/~dario/. Acesso em jun. 2008.

ARENALES, Marcos.; ARMENTANO Horacio.; MORABITO Reinaldo.; YANASSE Horacio, **Pesquisa Operacional.** 1ª ed. Rio de Janeiro: Elsevier, 2007.

AUDACES. **Empresa de desenvolvimento de software para soluções para o segmento de confecções.** Disponível em: http://www.audaces.com/novo/pt/home/. Acesso em dez. 2009.

BERNARDI, Reinaldo; KANG, C, Tsen. **Aplicando a Técnica de Times Assíncronos na Otimização de Problemas de Empacotamento Unidimensional.** Dissertação de mestrado. Escola Politécnica da Universidade de São Paulo, São Paulo, 2001. Disponível em: http://www.comp.ufla.br/curso/ano2002/monografias_bcc_ufla_2002.html. Acesso em Set. 2004.

BOAVENTURA, Netto, **P.O. Grafos: Teoria, Modelos, Algoritmos.** 4ª ed. São Paulo: Edgard Blucher, 2006.

CASTRO, M. P. **Reconstruindo-se a Função de Densidade Óssea Utilizando Triângulos de Bénzier em Dados Desestruturados para Aprimorar o Processo de Diagnóstico da Osteoporose.** Dissertação de mestrado. Universidade Federal de

Pernambuco. Fevereiro 2007. Disponível em:

http://www.bdtd.ufpe.br/tedeSimplificado//tde_busca/arquivo.php?codArquivo=2760. Acesso: 15/06/2009.

CGAL Editorial Board. **CGAL User and Reference Manual**. 3.5 edition, 2009. [WWW] [bibtex-key = cgal:eb-09 - bibtex file]

CGAL. ***Computational Geometry Algorithms Library.*** Disponível em: http://www.cgal.org/. Acesso: 10/06/2009.

CORMEN, Thomas H et al. **Algoritmos: teoria e prática.** Barueri: Campus, 2002. 916 p. ISBN 85-352-0926-3.

DANTZIG, G. B. **Discrete-Variable Extremum Problems**. Operations Research: Vol. 5, No 2, April 1957, p 266-288.

DIESTEL, Reinhard. **GraphTheory (3ª version)**. New York: Springer – Verlag Heidelberg, Eletronic Edition 2005. Disponível em: http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/index.html ou http://www.freebookcentre.net/ComputerScience-Books-Download/Graph-Theory,-3rd-Edition-(R.-Diestel).html. Acesso: 03 Feb. 2009.

DYCKHOFF, Harald; FINKE, Ute. **Cutting and Packing in Production and Distribuition: Typology and Bibliography.** Heidelberg: Springer-Verlag Co., 1992.

EOCAPLAN. **Empresa de distribuição e montagem de ambientes modulares.** Disponível em: http://www.eocaplan.com.br/. Acesso em nov. 2008.

EUCATEX. **Empresa fabricante da matéria prima para montagem de ambientes modulares.** Disponível em: http://www.eucatex.com.br/. Acesso em dez. 2009.

FERNANDES, H. Vitor**. Grafos e Aplicações. Grafos Planares.** Disponível em: http://ferrari.dmat.fct.unl.pt/personal/vhf/ga2003-2004/ga04cap08webfull.ps.gz. Acesso: 29 jan. 2009.

FIGUEIREDO, L. H & CARVALHO, P. C. Paulo. **Introdução à Geometria Computational.** 18° Colóquio Brasileiro de Matemática, IMPA, 1991, texto atualizado em 2005. Disponível em: http://w3.impa.br/~lhf/cursos/gc/. Acesso: 08 jan. 2009.

FLATO, E; HARPERIN, D; HANNIEL, I; NECHUSHTAN, O; EZRA, Eti. **The Design and Implementation of Planar Maps in CGAL.** Artigo. Outubro 2000. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.9140. Acesso: 14/08/2009.

GIEZEMAN, G; WESSELINK, W. **2D Polygons**. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.5 edition, 2009. [WWW] [bibtex-key = cgal:gw-p2-09 - bibtex file]

HANNIEL, I. **The Design and Implementation of Planar Arrangements of Curves in CGAL.** Dissertação de mestrado. Dezembro 2000. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.7241. Acesso: 14/08/2009.

KETTNER, L. **Halfedge Data Structures**. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.5 edition, 2009. [WWW] [bibtex-key = cgal:k-hds-09 - bibtex file]

KOERICH, L. Alessandro. **Fundamentos da análise da eficiência de algoritmos:**

Notação Assintótica. Disponível em:

http://dsc.ufcg.edu.br/~dalton/cursos/edados/NotacaoEAnaliseAssintotica.pdf. Acesso: 20 set. 2004.

OLIVEIRA, G. A. **Implementação do Plano Projetivo Orientado na Biblioteca CGAL. Dissertação de mestrado.** Universidade Estadual de Campinas. Dezembro de 2004. Disponível em: http://www.ic.unicamp.br/~rezende/T2Viewer/Oliveira.pdf. Acesso: 15/06/2009.

O'ROURKE, J. **Computational geometry in C.** Cambridge: Cambridge University Press 1994.

PIERRE, A. ;FABRI A. ;FOGEL F. **Couse notes about CGAL**. SIGGRAPH 2008. Los Angeles – Califórnia – USA. Disponível em: http://www-sop.inria.fr/members/Pierre.Alliez/ ou http://www.siggraph.org/s2008/. Acesso: 03/07/2009.

POIN, S. **Generic programming and CGAL**. http://www-sop.inria.fr/members/Monique.Teillaud/CGAL/. Acesso: 03/07/2009.

POLÍGONO aberto. In: Wikipédia: a enciclopédia livre. Disponível em: http://pt.wikipedia.org/wiki/Polígono. Acesso: 15 out 2009.

POLIEDRO aberto. In: Wikipédia: a enciclopédia livre. Disponível em: http://pt.wikipedia.org/wiki/Poliedro. Acesso: 15 out 2009.

RANGEL, S. **Introdução a construção de modelos de otimização linear inteira**. **Minicurso XXVIII CNMAC**. Disponível em:

http://www.dcce.ibilce.unesp.br/~socorro/mini_cnmac/. Acesso: 14 jan. 2009.

RZSISTEMAS. **Empresa de desenvolvimento de software para soluções CAD para o segmento de confecções.** Disponível em: http://www.rzsistemas.com.br/int/rzcadtextil.php?txt=1. Acesso em dez. 2009.

SULMODULOS. **Empresa de distribuição e montagem de ambientes modulares.** Disponível em: http://www.sulmodulos.com.br/index.htm. Acesso em dez. 2009.

SWARTZFITTER, J. L. **Grafos e algoritmos computacionais**. Rio de Janeiro: Campus, 1988.

YANASSE, H.H.; FURTADO, J.C. at Al. **O Problema de Corte e Empacotamento e Aplicações Industriais.** 2ª Oficina nacional de PCE. XX Congresso Nacional de Matemática Aplicada e Computacional. Gramado, 1997.

TCAD XP. **Componente para desenhos CAD.** Disponível em: http://www.codeidea.com/. Acesso em abr. 2008.

TEILLAUD, M. **Diversos materiais sobre CGAL**. http://www-sop.inria.fr/members/Monique.Teillaud/CGAL/. Acesso: 03/07/2009.

WAZLAWICK, R. **Metodologia de pesquisa para Ciência da Computação.** 1. ed. Rio de Janeiro: Elsevier, 2009.

WEIN, R; FOGEL, E; ZUKERMAN, B; HALPERIN, D**. 2D Arrangements**. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.5 edition, 2009. [WWW] [bibtex-key = cgal:wfzh-a2-09 - bibtex file].

ZONTA, Tiago; OLIVEIRA, L.I. **Comparações entre Formas de Resolução do Problema do Corte Unidimensional.** I Congresso de Lógica Aplicada à Tecnologia. São Paulo - SP, p.751-755. Out 11-15, 2000.

ZONTA, Tiago; OLIVEIRA, L.I. **Sistema CAD para edição de paredes de divisórias (DIVCAD).** In: IV Seminário de Iniciação Científica, 2000, Balneário Camboriú. Anais do IV Seminário de Iniciação Científica. , 2000.

ZONTA, Tiago; COSTA, J.I; OLIVEIRA, L.I; MAIA, L.F.J. **Inteligência Computacional Aplicada a Resolução do Problema do Corte Unidimensional.** Revista eletrônica de Sistemas de Informação. , 2005.