Modelação Geométrica com Restrições

Fábio Miguel Gonçalves Pinheiro
fabiopinheiro@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa

Resumo A modelação através da especificação de restrições geométricas permite ao modelador expressar-se diretamente no modelo. As ferramentas para modelação geométrica oferecem o máximo de funcionalidade possível e, cada vez mais, as novas funcionalidades estão relacionadas com a especificação de restrições geométricas, permitindo assim ao utilizador impor restrições e estabelecer relações entre as entidades geométricas. Este documento apresenta o estado da arte na área de restrições geométricas e oferece uma proposta para um sistema que permitirá a modelação geométrica através da especificação e satisfação de restrições geométricas. Foca-se ainda no problema de restrições geométricas, como é que estas podem ser resolvidas e quais as ferramentas que permitem ao utilizador especificá-las.

Palavras-chave:
Desenho Assistido por Computador; Modelação Geométrica; Restrições Geométricas; Satisfação de Restrições

1 Introdução

A modelação geométrica requer a especificação da localização, orientação e dimensão das entidades geométricas. Contudo, é frequente o modelador não conhecer essas propriedades *à priori*, mas conhecer outras que possam ser usadas para restringir e resolver essas propriedades. Por exemplo, o modelador pode restringir entidades geométricas através da especificação de relações com outras entidades, essas relações podem ser incidência, tangência, paralelismo, perpendicularismo, distância, volume, entre outras. Essas restrições têm de ser resolvidas de forma a descobrir a localização, orientação e dimensão das entidades geométricas. Atualmente os modeladores fazem este processo manualmente o que se torna muito moroso.

As restrições geométricas também podem ser usadas à posterior. Isto é, o conjunto de restrições geométricas pode ser usado para validar automaticamente um modelo geométrico. Por exemplo, por lei, as escadas não podem ter uma inclinação maior do que um determinado limite, e cada degrau tem que ser menor do que uma determinada altura. Todas essas restrições podem ser verificadas automaticamente por um sistema, detetando assim infrações permite informar de imediato o utilizador caso o conjunto das mesmas esteja a ser violado.

Permitir a modelação através da especificação de restrições geométricas é muito útil, pois possibilita que o modelador se expresse diretamente no modelo sem passar pelo demorado processo de encontrar a localização, orientação e dimensão das entidades geométricas.

A titulo de exemplo, consideremos que temos de desenhar uma figura com quatro pontos (A, C, M e T), três linhas ($\overline{AC}$, $\overline{AT}$ e $\overline{MT}$) e uma circunferência, descritos da seguinte maneira: a circunferência está centrada em C; a linha $\overline{AT}$ é tangente à circunferência e passa no ponto A; o ponto M é a interseção entre a linha $\overline{AC}$ e a linha perpendicular a esta que passa no ponto T. Apesar de todas estas especificações, o modelo geométrico continua ainda com alguns graus de liberdade, tais como: os pontos ainda não têm a sua localização exata embora existam relações entre eles, o raio da esfera não está definido e existem duas possíveis tangentes se o ponto A não estiver contido na circunferência. Contudo, pode-se inferir que o raio do círculo não pode ser maior que a distância entre A e C, uma vez que o ponto A tem de estar fora da circunferência (centrado em C) para que a tangente possa existir. Uma possível solução destas especificações está ilustrada na Figura 1.

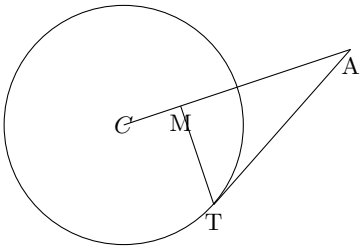


Figura 1: Modelo geométrico que satisfaz as especificações descritas no exemplo inicial.

Das especificações acima, não é possível concluir a localização final dos pontos. Isto é um problema, pois a maioria das ferramentas de modelação geométrica requerem essas informações. Por exemplo, para desenhar uma linha é necessário a localização dos pontos inicial e final da linha. Desta forma é difícil modelar as especificações acima, uma vez que o utilizador tem que se preocupar em calcular a localização desses pontos. Este processo é cansativo e requer a resolução de problemas geométricos e matemáticos potencialmente complexos. Todavia, o processo de modelação pode ser melhorado usando uma ferramenta adequada que suporte a especificação das restrições geométricas na criação do próprio modelo.

Como segundo exemplo, considere-se a criação de um pentágono tal como descrito no documento [1], através do método de construção geométrica tradicional, ilustrado na Figura 2. O método começa a partir dum dos lados do pentágono (o segmento de reta $\overline{AB}$ da figura), usando a régua e o compasso para encontrar interseções e assim construir o resto da figura. A utilidade deste tipo de raciocínio é subestimado pelas aplicações de desenho assistido por com-

putador (CAD - "computer-aided design"), o que faz com que o utilizador gaste o seu tempo a pensar como modelar o problema. No entanto, uma modelação declarativa através da especificação de restrições geométricas poderia facilitar todo o processo de modelação, exprimindo a intenção do desenhador diretamente no modelo geométrico.

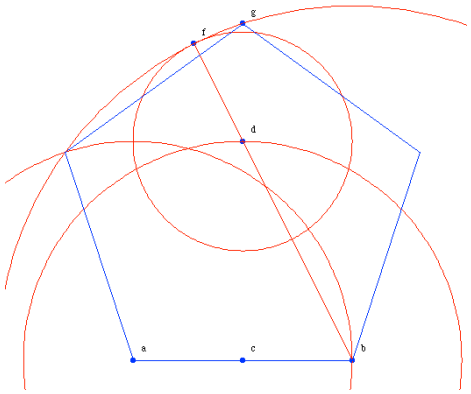


Figura 2: Construção de um pentágono com régua e compasso[1, Figura 1].

Um outro aspeto a cossiderar é a crescente procura por ferramentas que permitem a criação de algoritmos para desenho generativo. Em arquitetura os projetos estão a ficar cada vez mais sofisticados, tanto a nível de tamanho como de complexidade, por isso o custo de pequenas alterações através dos métodos tradicionais está a ficar incomportável. Assim, os grandes ateliers de arquitetura começaram a experimentar usar programação para modelar os edifícios, o que cria a necessidade de os arquitetos terem conhecimentos de programação e, por isso, algumas universidades já incluem uma cadeira de programação no curso de arquitetura. Este fato tem levado a um crescimento da procura por ferramentas de programação especializadas em modelação geométrica e desenho generativo que permitem ao utilizador gerar grandes quantidades de geometria de forma algorítmica.

Uma dessas ferramentas é o Rosetta, um ambiente de programação que permite o uso de diferentes linguagens de programação ("front-end") em diversas aplicações de CAD ("back-end"). Esta aplicação tem como objetivo libertar os utilizadores de estarem presos a uma família de aplicações CAD, facilitando assim a portabilidade. Infelizmente, o Rosetta, tal como a maioria das ferramentas deste tipo, não possui mecanismos para especificação de restrições geométricas. [2,3]

Este documento analisa várias linguagens de programação que suportam modelação geométrica através da especificação de restrições geométricas, entre estas linguagens encontra-se: Eukleides [4], Tikz [5], ThingLab [6], GeoSolver [7] e Grasshopper [8]. Estas linguagens de programação foram criadas com objetivos e finalidades distintas, embora todas elas tentem dar ao utilizador o máximo

de funcionalidades, para que deste modo o utilizador possa descrever o modelo geométrico fiel ao seu modelo mental sem que tenha de resolver restrições.

O documento tenta ainda responder às seguintes questões:

- O que é uma restrição geométrica?
- Como resolver um conjunto de restrições e quais as principais dificuldades?
- Quais são as ferramentas existentes que permitem especificar restrições geométricas a fim de facilitar o processo de modelação geométrica?
- Como é efetuada a interação entre o utilizador e essas ferramentas?

Esta secção introduziu o problema e os objetivos do trabalho. A secção 2 apresenta os conceitos relacionados com o problema de satisfação de restrições, foca-se na resolução de restrições geométricas e na importância da exatidão dos resultados e no fim analisa ferramentas mais adequadas para a resolução do problema em questão. Na secção 3 formalizamos os objetivos do trabalho. Na secção 4 é apresentada a arquitetura do sistema a ser construído. Na secção 5 é descrita a metodologia que será utilizada para avaliar o trabalho a realizar. Por último, na secção 6 são apresentadas as conclusões.

2 Objetivos

O objetivo desta tese é criar um sistema que permita ao utilizador fazer modelação geométrica através da especificação de restrições geométricas numa linguagem de programação. Temos o objetivo secundário de integrar o sistema criado com o projeto Rosetta, a fim de estender o poder descritivo da modelação geométrica do Rosetta com a especificação de restrições geométricas.

Tencionamos estender uma linguagem de programação para que o utilizador possa descrever restrições e fazer consultas às entidades geométricas do modelo, enquanto o cria. Devido ao objetivo secundário, assumimos que a maioria dos utilizadores são arquitetos com reduzidos conhecimentos de programação, por isso a linguagem de programação tem de ser simples, de fácil preceção para não requerer vastos conhecimentos de programação, mas tem que ter um poder descritivo suficiente para que seja útil.

Escolhemos integrar o projeto Rosetta para que este nos forneça toda a funcionalidade de modelação geométrica já oferecida peloas ferramentas de CAD, com o objetivo de nos abstrair desse género de modelação geométrica e para que nos possamos então focar em oferecer ao utilizador a possibilidade de usar as restrições geométricas na criação do modelo. Para além disto, ganhamos também uma forma de demonstração visual das capacidades do sistema criado.

O Rosetta servirá ainda como método de avaliação do sistema a ser criado, para que desta forma possamos ter um método de comparação entre a modelação geométrica com recurso às restrições geométricas e modelação geométrica usual oferecida pela maioria das aplicações.

3 Estado da Arte

Esta secção está dividida em duas subsecções, a primeira aborda os conceitos gerais que envolvem as restrições geométricas, a segunda aborda as ferramentas de software existentes relacionadas com o tema.

3.1 Conceitos Gerais

As restrições geométricas são limitações colocadas em entidades geométricas e entre elas. Dependendo do conjunto de restrições usadas no modelo geométrico este pode ter diversos graus de liberdade. Se o conjunto de restrições estiver totalmente restringido este tem apenas uma solução possível, ou seja, a localização, orientação e dimensão das entidades geométricas têm valores fixos.

As restrições geométricas podem ser expressas matematicamente por um sistema de equações, por exemplo, a restrição que especifica que o ponto $A = (x_a, y_a, z_a)$ está mais perto da origem que o ponto $B = (x_b, y_b, z_b)$ pode ser representada em notação matemática por uma inequação (Formula 1) que diz que a distância do ponto A à origem têm que ser menor que a distância do ponto B à origem.

$$\sqrt{x_a^2 + y_a^2 + z_a^2} < \sqrt{x_b^2 + y_b^2 + z_b^2} \tag{1}$$

As propriedades de uma entidade geométrica são um conjunto de características da própria entidade. Por exemplo, um cone tem as seguintes propriedades: raio da base, altura, localização do vértice de topo, localização do centro da base, direção do cone, volume, área de superfície, área lateral, área da base, entre outras. Muitas das propriedades estão relacionadas entre elas, por exemplo, a área de superfície é igual à soma da área lateral e da área da base, a direção do cone pode ser obtido através da localização do vértice de topo e do centro da base. Desta forma a mesma entidade geométrica pode ser descrita através de diferentes conjuntos de propriedades. Por exemplo, uma esfera é habitualmente descrita através do raio e centro mas pode ser descrita através de quatro pontos de superfície não colineares.

Problema de Satisfação de Restrições (CSP - "Constraint Satisfaction Problem") O CSP é um problema matemático (definição 1) que consiste num conjunto de objetos cujo estado tem que satisfazer todas as restrições, ou seja, consiste em encontrar um valor para cada variável (conjunto 'V' da definição 1), de entre os valores associados a essas variáveis (conjunto 'D' da definição 1), e onde as restrições (conjunto 'C' da definição 1) especificam que determinados valores não podem ser usados em conjunto.

Definição 1 *O CSP é o triplo P =(V,D,C) [9,10] onde:*

- ***V*** *= v_1, ..., v_n é o* ***conjunto de variáveis****.*
- ***D*** *= D_1, ..., D_n é o* ***domínio das variáveis****, isto é, D_i representa os valores que estão associados a v_i e que este pode assumir.*

- $\boldsymbol{C} = c_1, \ldots c_m$ *é o* ***conjunto de restrições****, onde cada* $c \in C$ *é um subconjunto do produto cartesiano* $\Pi_{i \in V_c} D_i$ *e* $V_c \subseteq V$ *é o conjunto de variáveis em c.*

O CSP é um problema altamente combinatorial e habitualmente da classe NP-completo.[1] [9]

Grau de Liberdade O grau de liberdade (DoF - "Degree of Freedom") de um CSP refere-se ao número de possíveis soluções do CSP. Um CSP pode estar demasiado restringido (Over-Constrained), pouco restringido (Under-Constrained) ou totalmente restringido (Well-Constrained), contudo não é óbvio saber à partida qual é DoF de um conjunto de restrições. Considere os seguintes exemplos, onde queremos calcular o valor da variável 'x', que existe num espaço unidimensional e está restringida pelas seguintes inequações:

- $x > 1 \wedge x < 1 \Rightarrow$ (Over-constrained) não existe solução para o CSP.
- $1 < x < 2 \Rightarrow$ (Under-constrained) existem inúmeras soluções para o CSP.
- $x \geq 1 \wedge x \leq 1 \Rightarrow$ (Well-constrained) existe uma só solução para o CSP.

Generalização do CSP Existem ainda alguns problemas mais genéricos, tais como o problema de otimização de satisfação de restrições (CSOP - "Constraint satisfaction optimization problem") e o problema de satisfação parcial de restrições (PCSP -"Partial constraint satisfaction problem"). O CSOP é um problema de otimização, cujo objetivo é encontrar a solução ou soluções que tenham um custo mínimo. Um exemplo típico é o problema de calendarização com o mínimo de horas livres. O PCSP é mais generalizado que o CSOP, divergindo deste quando o DoF é "under-constrained" ou "well-constrained," mas caso seja "over-constrained" o objetivo é encontrar a solução parcial que satisfaça o maior número de restrições possíveis. Um exemplo típico é o problema de máxima utilidade.

Domínio do CSP Quando falamos em domínio de um CSP estamos a referir ao domínio das variáveis (conjunto 'D' da definição 1), ou seja, ao conjunto de valores que cada variável pode assumir. A grande maioria da investigação efetuada sobre CSP assume que o conjunto de valores é finito. Por esta razão, quase todos os algoritmos para resolver CSP funcionam com essa condição. Para além disso, é bastante mais simples do que o caso infinito. Um algoritmo simplista para resolver estes CSP seria o de tentar todas as possíveis combinações de valores entre as variáveis até achar uma solução. Os algoritmos que tentam tratar os CSP com variáveis que podem assumir infinitos valores têm duas grandes condicionantes: a primeira é exigir muito mais poder computacional para tentar

[1] **NP-completo** é a classe de problemas que são ao mesmo tempo NP (dada uma solução, é verificável em tempo polinomial) e NP-difícil (um problema de decisão 'H' é NP-difícil, quando para qualquer outro problema NP 'L', existe uma redução de 'L' para 'H' em tempo polinomial)

resolver o CSP; a segunda é não serem capazes de garantir que não existe solução ou mais soluções que as encontrou. Podemos ainda subclassificar os CSP com domínio infinito em discreto (por exemplo, $x \in \mathbb{Z}$) e contínuo (por exemplo, $x \in [0, \pi]$).

Técnicas para Resolver CSP Em [11,12] são apresentadas cinco metodologias diferentes para resolver os CSP, estas são:

- Propagação local de restrições, esta metodologia representa o CSP num grafo não direcional que usa para propagar restrições geométricas, contudo o grafo não pode conter ciclos.

- Resolução numérica de restrições, esta metodologia representa o CSP por um sistema de equações algébricas e tenta encontrar soluções. Os algoritmos usados são capazes de lidar com grandes sistemas não-lineares mas assumem que o CSP está totalmente restringido e que cada variável pode apenas assumir um conjunto finito de valores (conjunto 'D' da definição 1).

- Manipulação simbólica, esta metodologia representa o CSP por um sistema de equações e usa métodos de manipulação simbólica para os resolver. Esta técnica pode concluir que a solução existe mas ser incapaz de a encontar, ou necessitar de tempo e memória exponencial, conforme a complexidade do CSP.

- Procura heurística, esta metodologia representa o CSP por fatos e regras que utiliza para criar um teste procedimental para descobrir soluções, semelhante ao Prolog, contudo os algoritmos usados são pouco escaláveis e desta forma são incapazes de lidar com grandes sistemas de restrições [6].

- Simplificação de CSP, esta metodologia é composta por duas fases e representa o CSP por um grafo. Na primeira fase as dependências entre os nós do grafo são analisadas para que o problema possa ser decomposto em vários problemas mais simples chamados clusters, cuja solução do problema é a junção das soluções desses clusters. Na segunda fase os clusters são formados e resolvidos individualmente usando outras técnicas (resolução numérica de restrições e manipulação simbólica).

A maior parte da investigação efetuada na resolução de CSP foca-se nos problemas mais simples, onde só existe uma solução e onde as variáveis podem assumir um conjunto reduzido de valores, muitas das vezes valores booleanos. A investigação efetuada nos problemas mais complexos usa algoritmos otimizados para o contexto do CSP em questão, parando procuras que, à partida, sabe-se que não fazem sentido no contexto do problema, reduzindo assim o número de possíveis caminhos da procura da solução. Porém, isto faz com que as técnicas usadas não sejam genéricas.

Robustez e Exatidão A falta de robustez ou exatidão é um problema bem conhecido na área de modelação geométrica [13]. Isto acontece porque muitos dos algoritmos de modelação geométricas são descendentes de algoritmos de computação gráfica, onde o tempo de computação é mais importante do que a fidelidade do modelo geométrico. No entanto, na modelação geométrica, os resultados podem ser reutilizados inúmeras vezes para cálculos posteriores, enquanto na computação gráfica os resultados são usados essencialmente para visualização. A maioria dos algoritmos de modelação geométrica são implementados com uma precisão finita, geralmente a aritmética de vírgula flutuante da máquina tornando-o dependente da especificação do software e do hardware.

Um exemplo típico é o caso de descobrir se duas retas são ou não paralelas. Para responder a esta questão compara-se o declive de cada uma das retas. Infelizmente se o declive for representado por um "float," o seu cálculo envolve erros de representação e arredondamentos. Por esta razão, as ferramentas podem afirmar que duas retas não são paralelas, embora teoricamente elas sejam, devido a discrepâncias mínimas entre os valores do declive, ou vice-versa. O que algumas aplicações, como o AutoCAD, fazem é assumir que as retas não paralelas têm declives significativamente diferentes, ou seja, duas retas são paralelas se a diferença entre os declives for inferior a um determinado valor suficientemente pequeno.

Mesmo usando bibliotecas como o CGAL [14] que tentam fornecer algoritmos robustos para modelação geométrica, é necessário usar boas práticas para evitar o uso de números irracionais. Por exemplo, acima apresentámos a restrição do ponto $A = (x_a, Y_A, z_a)$ estar mais perto da origem que o ponto $B = (x_b, y_b, z_b)$, e para descrever a restrição comparamos as distâncias de cada ponto à origem (Formula 1). Infelizmente isto implica o cálculo de raízes quadradas que iram produzir resultados inexatos. Assim, o que deveríamos ter sido feito era comparar não a distância mas o quadrado da distância (Formula 2).

$$x_a^2 + y_a^2 + z_a^2 < x_b^2 + y_b^2 + z_b^2 \tag{2}$$

3.2 Feramentas de Restrição Geométrica

Existe uma crescente procura de linguagens de programação que permitem fazer desenho generativo. Contudo, poucas oferecem a possibilidade de especificar restrições geométricas, e quase todas elas necessitam de conseguir resolver a restrição no momento em que é especificada. Nesta subsecção vamos discutir algumas das ferramentas de software e linguagens de programação que oferecem capacidades de modelação através da especificação de restrições geométricas. Infelizmente a resolução de CSPs é um problema complexo e existe pouca investigação que o aplique à modelação geométrica. Por isso, alargámos a nossa procura de ferramentas e aplicações para o que está a ser atualmente usado para satisfazer as necessidades cobertas pelo nosso tema. Por isso analisámos aplicações de CAD. Pode ser encontrado em [15] uma classificação destas. Estas aplicações disponibilizam o máximo de funcionalidade que conseguem, incluindo

algumas restrições geométricas, e muitas destas aplicações disponibilizam ferramentas e APIs para que possam interagir com outras aplicações. Isto leva a que existam ferramentas que ofereçam parte da funcionalidade das aplicações CAD, possibilitando assim ao utilizador fazer modelação geométrica através de uma linguagem de programação. Entretanto a modelação geométrica disponibilizada é muito básica e a especificação de restrições geométricas é reduzida ou inexistente. Isto é, enquanto numa aplicação CAD existem diversas interfaces gráficas especializadas para ajudar o utilizador a criar circunferências a partir de retas tangentes, pontos de superfície, entre outras, na API oferecida pela aplicação existe reduzidas maneiras de criar circunferências, nomeadamente com o centro e o raio. A tabela 1 apresenta as características mais interessantes de algumas destas ferramentas. A maioria das ferramentas requer conhecimentos de programação, sendo pouco adequadas para quem se inicia na programação. Isto constitui uma barreira inicial que faz com que muitos arquitetos desistam à partida.

A tabela 2 apresenta ferramentas que permitem ao utilizador fazer modelação geométrica através de uma descrição do modelo com recurso a especificação de restrições geométricas. Contudo a maior parte destas ferramentas necessitam que a restrição seja imediatamente resolvida assim que é descrita. Para além disso tem uma interface gráfica que impossibilita a criação de procedimento automatizados para gerar geometria. Na introdução demos o exemplo das escadas, que tinham de respeitar um conjunto de regras: por lei não podiam ter uma inclinação demasiado grande e cada degrau tem uma altura máxima. Estas restrições podem ser automaticamente validadas por um sistema de satisfação de restrições. As aplicações de Modelo de Informação da Construção (BIM - "Building Information Model") têm se tornado populares um pouco nesse sentido de obrigar o utilizador a respeitar um conjunto de regras enquanto modela o edifício. Nestas aplicações, os objetos têm muita semântica associada, não se limitando a forma geométrica, tentando assim englobar todo o ciclo de vida da construção do edifício. Por exemplo, dois tubos independentes não se podem intersetar fisicamente ou as janelas tem de estar numa parede e não podem estar no mesmo sítio que outras janelas ou portas, entre outras. Todavia, as aplicações BIM não são adequadas para modelação geométrica, uma vez que dificulta o trabalho dos arquitetos, porque obriga estes a definir logo o material dos objetos (modelo 'X' marca 'A') no processo de desenho. Estas especificações fazem variar o conjunto de regras que é necessário satisfazer. Por esta razão decidiu-se não analisar este género de ferramentas, uma vez que não representa grande interesse para o tema.

De seguida vamos analizar em pormenor algumas destas feramentas, mais concretamente a linguagem de programação Grasshopper da tabela 1, uma a aplicação para o CAD Rhinoceros 3D, pelo o fato de ser uma linguagem de programação visual, o que é uma característica interessante, porque reduz o impacto da barreira inicial de requerer comenhcimentos. Da tabela 2) escolhemos quatro ferramentas com características distintas e interessante para o nosso tema: Eukleides, GeoSolver, ThingLab e TikZ.

Grasshopper. É uma linguagem de programação visual, usada principalmente para a criação de algoritmos de desenho generativo, que depois geram geometria na aplicação CAD Rhinoceros 3D. [8]

No Grasshopper as funções são chamadas componentes e a maioria desses componentes geram geometria 3D que depois é exibida no Rhinoceros. Alguns dos componentes permitem ao utilizador encontrar intersecções [8] entre: duas linhas, uma linha e um arco, uma linha e um círculo, entre dois círculos, duas esferas, três planos, entre outros. Existe um componente diferente para realizar cada uma destas operações. No Grasshopper, os resultados dos componentes podem ser usados noutros componentes, desta forma existe também a preocupação com a robustez e validade dos resultados.

O inconveniente do Grasshopper é que a complexidade do código não é escalavel com a complexidade modelo, ou seja, o código fica cada vez mais difícil de compreender à medida que adicionamos complexidade ao modelo, como se pode observar na Figura 3. As linguagens de programação visual têm este inconveniente e, nestes casos, é mais produtivo usar uma linguagem de programação textual, principalmente para modelos complexos. Por outro lado, uma linguagem de programação visual é mais fácil e rápida para novos utilizadores apreenderem e começarem a usar. [16]

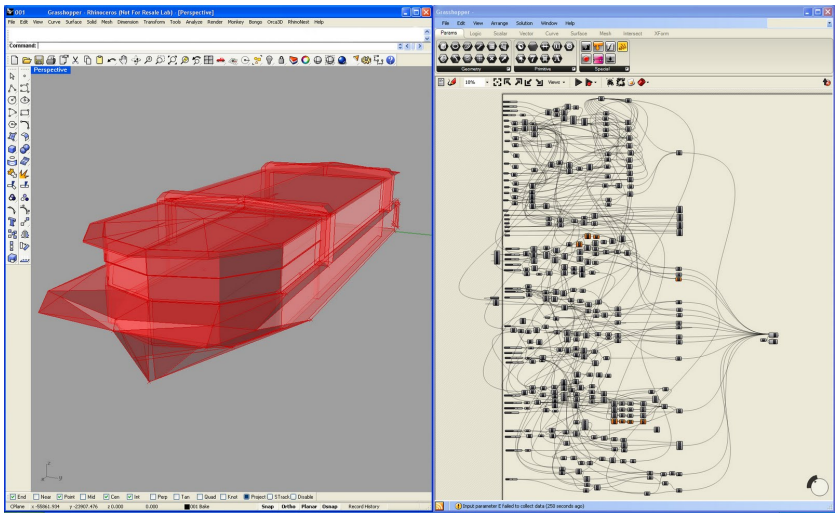


Figura 3: Grasshopper - Exemplo de código e respetivo resultado. Imagem retirada de `http://rhinocentre.blogspot.pt/` (Maio de 2015)

Eukleides. É uma linguagem de programação criada com o objetivo de descrever e gerar figuras bidimensionais. [4]

Os programas escritos em Eukleides consistem numa parte declarativa onde os objetos geométricos são definidos e noutra parte onde os objetos são desenhados. Para além disso, Eukleides é uma linguagem de programação completa,

com condições lógicas, estruturas iterativas e definição de funções, sendo capaz de lidar com os tipos básicos de uma linguagem de programação, mas também com entidades geométricas.

Listing 1.1: Eukleides - Código de um triângulo reto inscrito numa circunferência.

```
A B C right 5, 25deg
draw
  (A.B.C)
  circle(A, B, C)
end
```

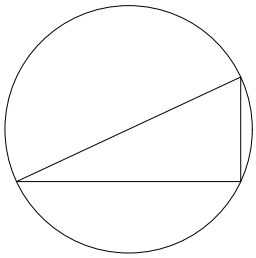


Figura 4: Triângulo reto inscrito numa circunferência.

A Figura 4 ilustra algumas das capacidades desta linguagem, esta foi gerada pelo código 1.1, escrito em Eukleides. A primeira linha do código é pouco compreensível ao leitor. Esta descreve três pontos que constituem os vértices de um triângulo reto no ponto 'B', em que o comprimento de $\overline{AB} = 5$ unidades e que o angulo $\widehat{BAC} = 25^o$. O restante código é apenas para desenhar o triângulo e a circunferência que passa pelos seus vértices.

ThingLab. É um ambiente de programação visual e foi pensado com o objetivo de ser um sistema computacional para a propagação de restrições. O utilizador especifica as restrições na interface gráfica que esta disponibiliza. As restrições são representadas como nós de um grafo que o sistema usa para propagar informação de umas partes para outras de forma a decidir se, e como, as restrições podem ser satisfeitas. [6]

Cada restrição tem um conjunto de métodos associados e cada um destes métodos representa uma forma distinta de satisfazer essa restrição. Como se pode observar na secção das restrições ("Constraints") do código 1.2, a restrição é satisfeita se um desses métodos devolver qualquer valor diferente de FALSO.

O ThingLab foi especialmente criado em cima da linguagem Smalltalk para permitir o estilo de programação "Data Flow," tornando-o particularmente interessante para o nosso tema. O "Data Flow" é um paradigma de programação para modelar o programa como uma série de ligações por onde a informação circula livremente. Estas ligações são funções que servem para transformar e propagar valores no CSP. A escrita das classes que representa restrições, requer conhecimentos intermédios de programação e uma boa compreensão tanto do funcionamento do sistema como de noções de matemática e de geometria. A

Listing 1.2: Classe da restrição do ponto intermédio de uma linha em ThinLag.

```
Class MidPointLine
  Superclasses
    Geometric Object
  Part Descriptions
    line: a Line
    midpoint: a Point
  Constraints
    midpoint = (line point1 + line point2)/2
    midpoint <-- (line point1 + line point2)/2
    line point1 <-- midpoint * 2 - line point2
    line point2 <-- midpoint * 2 - line point1
```

resolução de restrições usando o paradigma "Data Flow" é discutido em [17, chapter 3.3.5 – "Propagation of Constraints"].

Contudo, em ThingLab, todos os métodos alternativos para satisfazer as restrições têm de ser explicitamente descritos para que o paradigma de "Data Flow" funcione. No código 1.2 da classe da restrição do ponto intermédio de uma reta, existem quatro maneiras diferentes de satisfazer esta restrição, logo existem quatro métodos para a resolver. A restrição tem três campos de entrada/saída de valores (as duas extremidades da reta e o ponto intermédio). Em três dos métodos falta um desses campos. Estes métodos, quando são chamados (conforme o campo que falta), satisfazem sempre a restrição devolvendo o valor do campo em falta. O quarto método é chamado quando temos os três campos à partida, e neste caso, o método é uma função ($p_{medio} = (p_{iniciodelinha} + p_{fimdelinha})/2$) que devolve um valor booleano: caso esse valor seja FALSO, o teste falha significando que o conjunto de restrições não pode ser satisfeito. Para além disto esta técnica não funciona bem para CSPs com vários DoF ("under-constrained") e é pouco escalável porque o número de métodos chamados é exponencial com o número de restrições.

GeoSolver. É uma biblioteca de Python, usada para analisar e resolver restrições geométricas [7]. É capaz de tratar de restrições geométricas relativamente simples em modelos tridimensionais, tais como: a distância entre dois pontos, angulo entre três pontos, entre outros. O GeoSolver tenta encontrar soluções genéricas, ou seja, mesmo que o CSP tenha mais que uma solução (se for "under-constrained") a biblioteca tenta achar todas as soluções. [18,11]

O GeoSolver têm um *Workbench* com um interface gráfica (Figura 5), que pode ser usado para editar as restrições e ajudar o utilizador a escolher uma solução em particular. Esta ferramenta auxiliar tem uma abordagem interessante porque mostra ao utilizador, a decomposição em árvore do problema, de forma, a ajudar este a compreender os conflitos que possam existir entre as restrições. Por outro lado, é pouco conveniente mostra esta informação interna ao

utilizador, porque não é fácil para o utilizador normal compreende-la, uma vez que está diretamente relacionada com as técnicas usadas para resolver o CSP, e porque qualquer simples alteração no conjunto de restrições pode gerar uma decomposição completamente distinta.

Esta biblioteca é relativamente nova, havendo pouca documentação sobre a mesma. Para além disso pouco se sabe sobre a escalabilidade dos seus algoritomos e não é mencionado nenhuma vez o problema da robustez e exatidão dos resultados. Requer ainda que o utilizador tenha alguma destreza em programação e que compreenda a decomposição em árvore do problema.

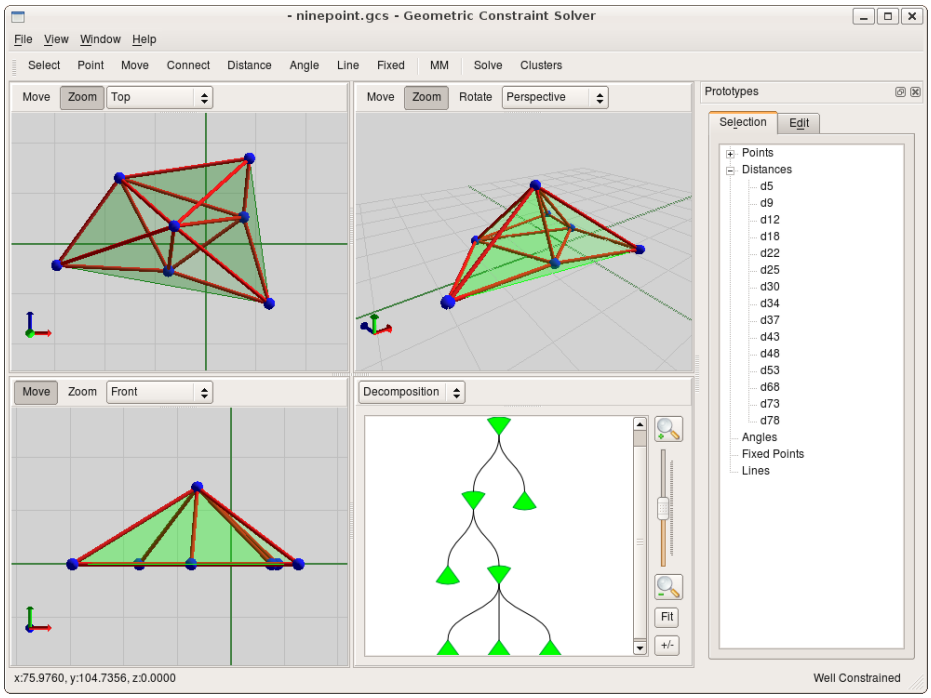


Figura 5: GeoSolver *Workbench* - Vista da solução e Vista o problema decomposto em árvore. Imagem retirada de [7] (Maio de 2015)

TikZ. É uma ferramenta usada para descrever figuras vetoriais num espaço bidimensional, através de descrições geométricas e algébricas. Os programas de TikZ são escritos em LaTeXcom uma sintaxe especial, invocando macros de TeX. O TikZ fornece uma sintaxe facilmente extensível com um elevado nível de abstração, enquanto a sua linguagem par, "PGF," fornece uma sintaxe de baixo nível. [5]

A Figura 1 é uma possível solução ao exemplo inicial e foi gerada a partir do código 1.3. Embora as coordenadas exatas dos pontos e o raio da circunferência

Listing 1.3: Código TikZ usado para gerar a Figura 1.

```
1 \begin{tikzpicture}
2   \coordinate [label=below:A] (a) at (5,3);
3   \node [circle,draw] (c) at (2,2) [minimum size=90pt] {$C$};
4   \coordinate [label=below:T] (t) at (tangent cs:node=c,point
        ={(a)}, solution=2);
5   \draw (a) -- (t);
6   \draw (c.center) -- (a);
7   \coordinate [label=below:M] (m) at ($(a)!(t)!(c)$);
8   \draw (t) -- (m);
9 \end{tikzpicture}
```

sejam desconhecidas na especificação do exemplo inicial. Estas têm que ser especificadas nos programas de TikZ, porque as restrições do programas em TikZ têm que estar "well-constrained," ou seja, só pode haver uma solução possível. Por isso nas linhas 2 e 3 do código é definido o valor do raio e a localização dos pontos. Para criar a reta $\overline{TM}$ (perpendicular à reta $\overline{AC}$ que passa por T), o ponto M é calculado usando a sintaxe "($(A)!(T)!(C)$)" (linha 7 do código), que descreve o ponto T projetado na reta que passa nos pontos A e C. Para fazer a tangente, o TikZ tem um sistema de coordenadas chamado "tangent" (linha 4 do código), que permite a computação do ponto tangente através da seguinte sintaxe: <node><point><number>. O campo "node" é o objeto geométrico onde queremos criar a tangente. O campo "point" é o ponto por onde a tangente deve passar. Finalmente o campo "number" é a solução que queremos escolher. Uma vez que existem duas soluções de tangentes à circunferência que passa pelo ponto, é necessário escolher uma em concreto.

O TikZ permite ao utilizador definir novos sistemas de coordenadas para satisfazer as suas necessidades usando o PGF como outras bibliotecas de matemática/cálculo. Pensar na tangente como um sistema de coordenadas é interessante e permite ao utilizador selecionar a solução pretendida caso exista mais que uma. No entanto, para os espaços tridimensionais a seleção da solução tem de ser mais sofisticada. No AutoCAD existe um vasto conjunto de operação para selecionar pontos 'interessantes', esta seleção é efetuada na própria figura através da interfaces gráfica. E embora funcione bem para desenho bidimensional é complicado e confuso para desenho tridimensional devido a não existir uma relação de dimenção espacial de um para um entre o desenho tridimensional e o monitor e rato. Contudo a ideia do sistema de coordenadas podia ser aplicado no espaço tridimensional. Por exemplo, imaginando que queremos a tangente entre um esfera e um ponto, como é óbvio existem infinitas soluções. Podemos então pensar nas soluções como sendo a superfície lateral de um cone, que contêm a esfera e cujo vértice do cone é o ponto onde passa a tangente. Assim, seleciona-se a tangente pretendida através de um ângulo, a sintaxe será semelhante à do TikZ com a diferenciação de, em vez de ser um número para selecionar a solução, seria um ângulo.

4 Arquitetura do Sistema

Esta secção apresenta a estratégia que será seguida durante o desenvolvimento do trabalho, direcionada aos objetivos de criar uma ferramenta que permita a modelação geométrica através da especificação de restrições e a sua integração com o projeto Rosetta.

O primeiro passo corresponde, à escolha da técnica mais adequada à resolução de restrições geométricas, num contexto da modelação geométrica aplicado à arquitetura. Aqui revolveu-se usar um modelo hibrido, entre a resolução de CSP e a utilização de funções de problemas concreto de geometria. Entre as metodologias de resolução de CSP, orientadas aos objetivos descritos, a metodologia de simplificação do CSP juntamente com a manipulação simbólica revelaram-se as mais adequadas. Pois neste contexto, trabalhamos num espaço bidimensional e tridimensional onde as variáveis podem assumir valores contínuos e a validade de alguns resultados é indispensável. Os modelos geométricos são por norma complexos, contendo muitas especificações, embora a maior parte das restrições geométricas possam ser agrupadas e resolvidas em grupos independentes.

O sistema vai consistir numa linguagem de programação declarativa, uma base de fatos, um avaliador e em dois núcleos de cálculo. A linguagem de programação será usada pelo utilizador para descrever restrições geométricas e fazer consultas ao modelo. A base de fatos servirá para gravar as especificações do utilizador, regras predefinidas de geometria e informações auxiliares. O avaliador será responsável por manter a base de fatos atualizada à medida que recebe as especificações do utilizador, e ainda, por analisá-la de forma a responder às consultas que o utilizador pretenda efetuar ao modelo. Nessa análise, o avaliador tenta descobri qual a melhor maneira de resolver e modelar num problema concreto para ser resolvido por um dos núcleos de cálculo.

Toda a interação entre os diferentes componentes do sistema está ilustrada na Figura 6.

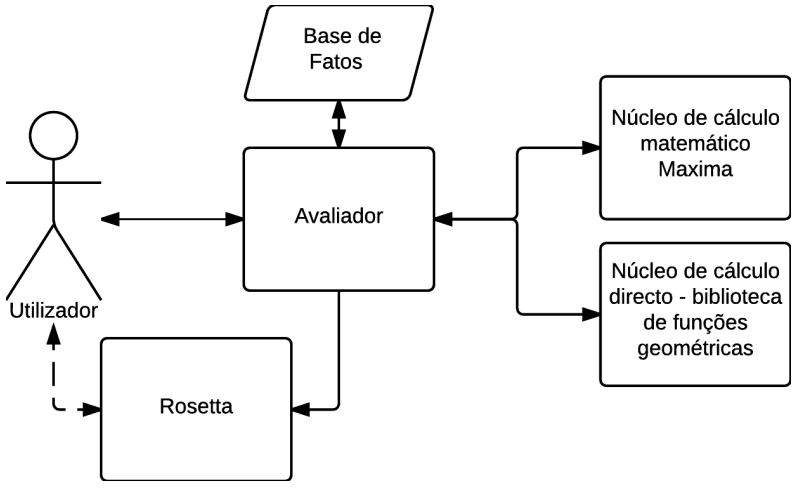


Figura 6: Arquitetura do Sistema - Interação entre componentes.

4.1 Linguagem de Programação

A interação do utilizador com o sistema é efetuada através da linguagem de programação que iremos criar. A linguagem será implementada como um módulo de linguagem do Racket de forma a proporcionar interoperabilidade com o Racket, o Rosetta e módulos de linguagem do Racket. Assumimos que o nosso utilizador típico será o utilizador típico do Rosetta, ou seja, um arquiteto com conhecimentos de programação que pretende fazer modelação geométrica. Portanto a sintaxe da linguagem de programação devera ser simples e intuitiva, de forma será uma linguagem do estilo declarativo e usara termos arquitetónicos. Pretendemos assim, que o processo de descrição seja semelhante ao raciocínio lógico usado na modelação geométrica semelhante às aplicações CAD.

Esperamos ainda que as funcionalidades mais úteis e desejadas pelos arquitetos sejam o espaço bidimensional, a especificação de entidades geométricas através das suas diferentes propriedades e o cálculo de interseções, paralelas, perpendiculares e tangentes, entre outras.

A escolha da linguagem Racket como a linguagem de programação base é devido à interoperabilidade que esta oferece e porque o Racket é uma linguagem simples usada para ensinar programação, mas ao mesmo tempo poderosa. Para além disso é também a linguagem base do Rosetta o que ajudará na integração com este.

4.2 Base de Fatos

Toda a informação descrita pelo utilizador é guardada em diversas estruturas de dados. Existe uma estrutura de dados genérica, que acomodará todas as formas geométricas. Esta é constituída por três campos: um identificador único da instância, um identificador do tipo da forma geométrica e uma lista de propriedades da forma geométrica utilizadas para descrever a instância.

Para cada forma geométrica existe um conjunto de funções e uma lista de todas as propriedades possíveis de associar. Das funções associadas aos tipos, existe um conjunto de funções genéricas a todos os tipos mas que são especializadas para cada tipo geométrico. Por exemplo a função para desenhar a forma geométrica através do Rosetta, saberá como chamar corretamente a API do Rosetta para o tipo da forma geométrica em questão. Existe também um conjunto de funções únicas de cada tipo geométrico que servirá para relacionar as propriedades geométricas. Por exemplo, estas relações podem ser, entre o volume de uma esfera e o seu raio, entre a localização de três pontos de uma circunferência e o centro ou centro da mesma. Existe um conjunto de funções para gerar equações a partir das propriedades, de forma a descrever as formas geométricas em notação matemática.

Existe ainda uma estrutura de dados para as variáveis que é constituída por três campos: um identificador único da instância, o valor da variável e informação auxiliar. O valor da variável pode estar ou não estar definido, podendo este ser um tipo numérico do Racket ou ser um conjunto de carateres em notação matemática. A informação auxiliar informa sobre a origem e finalidade da variável,

isto é, se foi o utilizador que a introduziu ou se foi gerada e onde é usada a variável.

Resumindo, estas estruturas da base de fatos serve para construir um grafo que relaciona todas as informações sobre as entidades (as especificações do utilizador com os conceitos predefinidos de geometria), para ajudar o avaliador a formular os CSPs de forma a responder às consultas.

4.3 Avaliador

O avaliador interpretará a sintaxe da linguagem de programação e será responsável por controlar todo o fluxo de informação dentro do nosso sistema, a comunicação com o Rosetta e a interação com o utilizador.

Pretendemos evitar computações desnecessárias, por isso, vamos retardar as propagações de valores no grafo de base de fatos até que seja estritamente necessário responder às consultas de utilizador. No estilo de programação "Data Flow", usado no ThingLab, a informação é propagada no grado assim que é recebida, enquanto no nosso sistema o grafo é usado para pesquisar quando é necessário e responder às consultas.

Para melhor perceber o motivo da linguagem de programação ser declarativa suponhamos o pseudo-code 1.4, onde todas as entidades são variáveis:

Listing 1.4: Exemplo de programação declarativa.

```
1 A = 0
2 B = A + 1
3 C + 7 = T + V
4 T = 10
5 C = B
6 V = -2
7 eval(B)
8 eval(V)
```

Note-se que 'A', 'B', 'C', 'T' e 'V' são variáveis, que as linhas 7 e 8 são consultas efetuadas ao modelo, e que as primeiras 6 linhas de código são especificações, onde o utilizador declara como quer o modelo geométrico. O avaliador guarda estas informações na base de fatos, onde vai construindo um grafo com as entidades e as relações entre as elas.

Repare que a variável 'B' tem múltiplas especificações. Quando o avaliador chega a linha 7 ele analisa a base de fatos começando pelo nó 'B' do grafo, este tem duas ligações com outros nós criados pelas linhas 2 e 5. Neste caso o avaliador pode optar por resolver as especificações das linhas 1 e 2 ou as especificações das linhas 3, 4, 5 e 6. De seguida responde ao utilizador e guarda na base de fatos o resultado, os cálculos intermédios e as informações sobre estes. Na consulta da linha 8 o avaliador vai a base de fatos e responde imediatamente ao utilizador.

4.4 Núcleos de Cálculo

Existem dois núcleos de cálculo, o de cálculo matemático, composto por um sistema de manipulação simbólica e computação algébrica, e o de cálculo direto, composto por uma biblioteca de funções para problemas geométricos bem conhecidos. Esperamos que núcleo de cálculo direto resolva a maior parte das que nessecidades dos arquitetos, e sempre que possível o sistema dará preferência à utilização deste núcleo.

O núcleo de cálculo direto é constituído por um conjunto de funções para resolver problemas geométricos concretos. Este núcleo de cálculo tem como objetivo otimizar a avaliação dos problemas geométricos mais populares, atravez da utilização de funções para resolver estes problemas geométricos. Evitando assim, a utilização do núcleo de cálculo matemático, porque este tem um custo computacional elevado e não garante que encontra a solução. Neste núcleo de cálculo cada problema geométrico será resolvido por uma função bem definida. Os problemas resolvidos por este núcleo de cálculo são por exemplo, o cálculo do centro de uma circunferência dado três pontos, o cálculo da intersecção de duas retas dado dois pontos de cada reta, entre muitos outros.

No núcleo de cálculo direto, uma consulta ao modelo é mapeado numa lista de funções, de seguida consultamos a base de fatos à procura dos argumentos destas funções, caso seja encontrado todos os argumentos duma função da lista, esta é chamada e o resultado é devolvido ao utilizador.

O núcleo de cálculo matemático serve dois propósitos. O principal propósito é resolver uma gama de problemas geométricos muito mais abrangente do que o núcleo de cálculo direto. Porque nunca iríamos conseguir imaginar, nem resolver no núcleo de cálculo direto, todas as possíveis situações que o utilizador possa descrever.

No núcleo de cálculo matemático, as consultas são transformadas em equações com variáveis livres, de seguida vamos à base de fatos converter para equações as entidades geométricas e as suas propriedades envolvidas nessa consulta. Todas estas equações são então enviadas para o módulo de manipulação simbólica e computação algébrica, na esperança que o módulo consiga resolver o sistema de equações em ordem às variáveis livres. Note-se que a eficácia do núcleo de cálculo matemático dependerá muito de três pontos cruciais:

- Da forma como as diferentes propriedades estão relacionadas no **nosso sistema** e como estas são convertidas para equações.
- Da capacidade que o **módulo de manipulação simbólica e computação algébrica** tem para resolver sistemas de equações.
- Das especificações que o **utilizador** forneça. Estas podem ser insuficientes e/ou estarem em contradição.

O outro propósito do núcleo de cálculo matemático é garantir a robustez e a exatidão dos resultados. Uma vez que o módulo de manipulação simbólica e computação algébrica vai simplificar e resolver equações em notação matemática, através de técnicas de manipulação simbólica, este vai evitar arredondamentos, garantindo assim a validade dos resultados.

Em contrapartida a utilização desta aplicação inclui converter as entidades geométricas envolvidas na consulta e as suas propriedades num sistema de equações, enviá-lo ao módulo de manipulação simbólica e computação algébrica, onde o sistema de equações é resolvido, os resultados são enviados de volta e a resposta é formular. Uma vez que toda esta computação tem um grande custo computacional, damos preferência a utilização do núcleo de cálculo direto sempre que possível e que não seja necessário resultados exatos. Esperamos que os

resultados exatos sejam principalmente só necessários quanto estamos a fazer comparações com valores semelhantes.

No processo de escolha do sistema de manipulação simbólica e computação algébrica comparámos diversas aplicações entre as quais: Maxima, Mathematica, Sage (System for Algebra and Geometry Experimentation). Estas aplicações têm capacidades e dificuldades muito semelhantes, em particular, todas elas conseguem resolver as mesmas classes de equações e têm dificuldades em resolver inequações.

A aplicação Maxima foi escolhida dentro de alguns possíveis sistemas de manipulação simbólica e computação algébrica, pelo fato de ser grátis, de software livre com o código fonte disponível sobre a licença GPL.[2] O fato do Maxima ter uma implementação em Lisp é um aspeto positivo porque nos permite usar a representação interna do Maxima na comunicação. Isto, pode a vir ser útil para guardar, modificar e reutilizar os cálculos intermédios, porque o Racket é uma linguagem derivada do Lisp com uma sintaxe semelhante.

Para melhor perceber como é que os problemas geométricos podem ser resolvidos no Maxima suponha, que queremos criar uma esfera cuja superfície passa pelos quatro pontos seguintes (3,2,1), (1,-2,-3), (2,1,3) e (-1,1,2). Este problema pode ser resolvido diretamente pelo método de W.H.Beyer [19, secção 4.18.1]. Contudo, se não tivemos um método direto na biblioteca de funções este caso, podíamos converter o cenário num sistema de equações para ser resolvido no Maxima. A superfície da esfera é definida pela seguinte equação $(x-a)^2+(y-b)^2+(z-c)^2=r^2$, desta forma podemos gerar quatro equações, uma para cada um dos pontos de superfície.

Listing 1.5: Maxima - circunferência definida por 4 pontos.

```
(%i1)> e1: (x-3)^2 + (y-2)^2 + (z-1)^2 = r^2 $
(%i2)> e2: (x-1)^2 + (y-(-2))^2 + (z-(-3))^2 = r^2 $
(%i3)> e3: (x-2)^2 + (y-1)^2 + (z-3)^2 = r^2 $
(%i4)> e4: (x-(-1))^2 + (y-1)^2 + (z-2)^2 = r^2 $
(%i5)> algsys ([e1,e2,e3,e4], [x,y,z,r]) ;
                24         16       4          3 sqrt(470)
(%o5) [[x = --, y = - --, z = --, r = - -----------],
                19         19       19             19
                    24         16       4        3 sqrt(470)
               [x = --, y = - --, z = --, r = -----------]]
                    19         19       19           19
```

O Maxima devolve duas soluções (Formula 3) para este sistema de equações, mas uma delas é invalida no contexto do problema, porque a variável 'r' representa o raio da esfera e o seu valor não pode ser negativo. Podia-se acrescentar a restrição que o raio tem que ser maior que zero ($r > 0$) mas isto é uma inequação

[2] **GPL** ou **General Public License** é uma licença que permite executar, modificamos e redistribuir todas as versões de um programa, certificando que continua a ser software livre para todos os seus utilizadores (acesso ao código-fonte é um pré-requisito para esta liberdade).

e como já referido os sistema de manipulação simbólica e computação algébrica tem dificuldades em resolver inequações. Desta forma é preferível resolver o sistema de equações e de seguida remover as soluções que não fazer sentido no contexto do problema.

$$\left[\left[x = \frac{24}{19}, y = -\frac{16}{19}, z = \frac{4}{19}, r = -\frac{3\sqrt{470}}{19}\right], \left[x = \frac{24}{19}, y = -\frac{16}{19}, z = \frac{4}{19}, r = \frac{3\sqrt{470}}{19}\right]\right] \quad (3)$$

5 Avaliação

O nosso objetivo é facilitar a modelação geométrica, dando ao utilizador a possibilidade de usar restrições para descrever o modelo. Por isso pretendemos avaliar a utilidade que o sistema vai trazer ao utilizador, a performance do sistema e a robustez dos resultados dos núcleos de cálculo.

A utilidade do sistema será avaliada através de testes com utilizadores, que consistirão em alguns cenários geométricos para modelar. Os utilizadores serão divididos em dois grupos independentes, um dos quais usará o Roseta e o outro grupo usará o sistema criado integrado no Roseta. Os cenários da experiência serão especificados em texto e descreverão modelos geométricos com vários níveis de complexidade e requisitos. Queremos usar problemas arquiteturais reais nos cenários, para que a linguagem usada na especificação dos cenários seja independente e não influenciar o modo como vai ser modelado. A métrica de avaliação vai ser o tempo necessário para a criação dos modelos.

Para comparar os dois núcleos de cálculo pretendemos gerar automaticamente vários modelos geométricos abstratos que possam ser resolvidos pelos dois núcleos de cálculo e desta forma comparar a performance de cada um deles. Os modelos gerados irão testar individualmente os diferentes problemas geométricos suportados pelo núcleo de cálculo direto. A métrica de avaliação vai ser o tempo de computação. Para analisarmos a robustez e a validade vamos comparar os resultados obtidos dos dois núcleos de cálculo com os valores esperados teoricamente. Podíamos ainda comparar a memória requisitada nas duas situações, mas esperamos que a diferença seja pouco significativa, com um gasto de memoria ligeiramente superior por parte do núcleo de cálculo matemático.

Na avaliação da utilidade do sistema para o utilizador esperamos que o tempo necessário para criar o modelo seja significativamente menor. Será ainda interessante olhar para o código criado no teste com utilizadores e analisar a reutilização do código para pequenas alterações nas especificações, ou seja, qual a quantidade de código que é necessario de alterar caso houvesse alterações nos valores da descrição ou nos requisitos.

Na comparação dos dois núcleos de cálculo esperamos que o núcleo de cálculo direto requisite menos tempo de computação que o núcleo de cálculo matemático.

Em contrapartida, em alguns nos testes, como o de paralelismo, de perpendicularidade, entre outros, esperamos uma pequena discrepância nos valores dos resultados devido aos arredondamentos. Essa discrepância será maior, quanto maior for a dependência entre os cálculos, ou seja, se houver uma grande cadeia de cálculos, onde os próximos cálculos usam os resultados dos anteriores.

Por fim, pretendemos avaliar se fomos ao encontro das necessidades dos arquitetos, para isso, vamos compara a percentagem de utilização entre os dois núcleos. Um vez que, queremos cobri a maioria dos cenários mais usados no núcleo de cálculo direto, esperamos que a sua utilização do seja muito superior a utilização do núcleo de cálculo matemático.

6 Conclusão

Neste documento abordámos o problema de satisfação de restrições aplicado à modelação geométrica e explicámos as vantagens em usar ferramentas que possibilitam a modelação geométrica através da especificação de restrição geométricas, e como estas ajudarão os arquitetos a construir edifícios complexos.

A comunidade de arquitetura está se a voltar para a programação de forma a gerar grandes quantidades de geometria, para que desta forma possam construir edifícios complexos através de algoritmos procedimentais que seguem especificações geométricas, incluído restrições geométricas. Os arquitetos tencionam assim reduzir o tempo e custo de efetuar alterações ao modelo inicial ou fazer experiências rápidas. Desta forma, começa-se a notar uma crescente procura por ferramentas que permitem fazer modelação geométrica através da especificação de restrições geométricas.

Definimos o problema de satisfação de restrições e apresentámos os conceitos relacionados com este. Dissemos que o CSP é um problema NP-completo e que a sua complexidade é muito influenciada pelo grau de liberdade e domínio das suas variáveis. Por isso, a maior parte sistemas para resolver CSP só trabalham com problemas bem definidos, onde só há uma solução e as variáveis têm um conjunto finito de valores que podem assumir. A maior parte da investigação existente sobre a resolução de CSP foca-se nestes problemas bem definidos por isso a técnicas usadas são genéricas, podendo ser aplicadas a CSP em diversos contextos. Em oposição, os outros CSPs são resolvidos por técnicas especializadas num determinado contexto, de forma a otimizar a procura de soluções, impedindo caminhos de procura que à partida sabe-se que não fazer sentido no contexto do problema, reduzindo assim o número de possível caminhos na procura da solução.

Abordámos o fato dos algoritmos de modelação geométrica serem descendentes dos algoritmos de computação gráfica, onde os requisitos são diferentes, trazendo assim o problema da falta de robustez e exatidão nos resultados. Principalmente devido aos arredondamentos efetuados pelas operações básicas da máquina. Por isso os algoritmos devem ser desenhados usando boas práticas, como a de evitar o uso de números irracionais uma vez que estes não são repre-

sentados fielmente. Desta forma intencionamos usar a notação matemática para representar os valores.

Apresentámos ferramentas que existem atualmente e que estão relacionadas com o tema. Abordou-se primeiro as ferramentas e APIs para as aplicações CAD que permitem aos arquitetos criar algoritmos procedimentais para fazer desenho generativo. Contudo o conjunto de capacidades oferecidas é bastante reduzido em comparação com a interface gráfica da aplicação, sendo quase inexistentes as capacidades que permitem fazer a especificação de restrições geométricas diretamente no modelo. Referimos que as aplicações BIM podiam ser usadas para validar restrições como as descritas nos regulamentos da construção, uma vez que o modelo tem muita semântica associada, contudo essa semântica tem que ser totalmente especificada na fase de desenho, como o tipo e marca do tijolo usado na cada parede, o que dificulta seriamente a projeção do edifício. No fim da secção apresentámos diversas ferramentas menos relacionados com arquitetura, mas que possibilitam ao utilizador a especificação de um maior número de restrições geométricas no processo de modelação, contudo estas restrições têm que ser resolvidas no momento que são especificadas.

Por fim, definimos os objetivos para o trabalho a ser realizado na tese e propusemos um sistema para ajudar o utilizador no processo de modelação através da especificação declarativa de restrições geométricas. Apresentámos a arquitetura do sistema proposto, fizemos ainda uma calendarização para o trabalho (Tabela 3) e propusemos um método de avaliação do sistema.

Referências

1. Harry Mairson. Functional geometry and the trait'e de lutherie. 1990.
2. António Menezes Leitão. Programação para arquitectura. Rosetta manual, 2012.
3. José Lopes and António Leitão. Portable generative design for cad applications. 2011.
4. Eukleides - A geometry drawing language. From Eukleides Web Page: `http://www.eukleides.org`. Accessed: 2015-05-20.
5. Till Tantau. Tikz & pgf: Manual for version 2.10, 2012.
6. Alan Borning. The programming language aspects of thinglab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 3(4):353–387, 1981.
7. GeoSolver. From GeoSolver Web Page: `http://geosolver.sourceforge.net`. Accessed: 2015-05-20.
8. P. Andrew and R. Isaa. *The Grasshopper Primer*, volume Second Edition. 2009.
9. O Le Roux, V Gaildrat, and R Caube. Constraint satisfaction techniques for the generation phase in declarative modeling. In *Geometric modeling: techniques, applications, systems and tools*, pages 193–215. Springer, 2004.
10. Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.
11. Rogier de Regt, Hilderick A van der Meiden, and Willem F Bronsvoort. A workbench for geometric constraint solving. *Computer-Aided Design and Applications*, 5(1-4):471–482, 2008.
12. Maurice Dohmen. A survey of constraint satisfaction techniques for geometric modeling. *Computers & Graphics*, 19(6):831–845, 1995.
13. Chee-Keng Yap. Towards exact geometric computation. *Computational Geometry*, 7(1):3–23, 1997.
14. Menelaos I Karavelas. Exact geometric and algebraic computations in cgal. In Komei Fukuda, Jorisvander Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software–ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 96–99. Springer Berlin Heidelberg, 2010.
15. Classification of CAD applications. Web Page: `http://academy.cba.mit.edu/classes/computer_design`. Accessed: 2015-06-01.
16. António Leitão, Luís Santos, and José Lopes. Programming languages for generative design: A comparative study. *International Journal of Architectural Computing*, 10(1):139–162, 2012.
17. Harold Abelson and Gerald Jay Sussman. Structure and interpretation of computer programs. 1983.
18. Hilderick A van der Meiden and Willem F Bronsvoort. A non-rigid cluster rewriting approach to solve systems of 3d geometric constraints. *Computer-aided design*, 42(1):36–49, 2010.
19. Daniel Zwillinger. *CRC Standard Mathematical Tables and Formulae*. CRC press, 32 edition, 2011.
20. Gus Lopez, Bjorn Freeman-Benson, and Alan Borning. Kaleidoscope: A constraint imperative programming language. In *Constraint Programming*, pages 313–329. Springer, 1994.

A Anexos

A.1 Ferramentas

Tabela 1: Lista de APIs e ferramentas de programação dos CAD.

Ferramentas	CAD[3]	Características
ActiveX Automation for nanoCAD[4]	nanoCAD	Linguagens JavaScript e VBScript
Antimony	Própria aplicação	Linguagem de programação visual e textual (Python); Excelente mecanismo de CSG.[5]
AutoCAD .Net API[6]	AutoCAD	Grande poder descritivo; Requer vasto conhecimentos de programação
AutoLisp[7]	AutoCAD	Dialeto de Lisp; Grande comunidade
Grasshopper[8]	Rhino	Linguagem de programação visual; Grande comunidade
ObjectARX[9]	AutoCAD	Grande poder descritivo; Requer vasto conhecimento de programação
Python scripting for Rhino[10]	Rhino	Linguagem Python; Fácil de aprender; Interligação com Grasshopper
RhinoScript[11]	Rhino	Linguagem VBScript
Rosetta [2,3]	diversas aplicações	Fácil de aprender; Termos arquiteturais; Suporta diversas linguagens de programação
Sketchup Ruby API[12]	Sketchup	Linguagem Ruby, Semelhante a lingua natural; Boa integração com o CAD
Visual Lisp[13]	AutoCAD	Linguagem de programação visual

[3] CAD - "computer-aided design" ou desenho assistido por computador. Pode ser encontrado em [15] uma classificação das aplicações CAD.
[4] `http://nanocad.com/page/Programming1/`
[5] CSG - "Constructive solid geometry" é um técnica usado na modelação geométrica de sólidos, através de operações binárias.
[6] `http://docs.autodesk.com/ACD/2010/ENU/AutoCAD%20.NET%20Developer%27s%20Guide/`
[7] `http://en.wikipedia.org/wiki/AutoLISP/`
[8] `http://www.grasshopper3d.com/`
[9] `http://en.wikipedia.org/wiki/ObjectARX/`
[10] `http://wiki.mcneel.com/developer/python/`
[11] `http://www.rhinoscript.org/`
[12] `http://www.sketchup.com/intl/en/developer/`
[13] `http://www.afralisp.net/visual-lisp/`

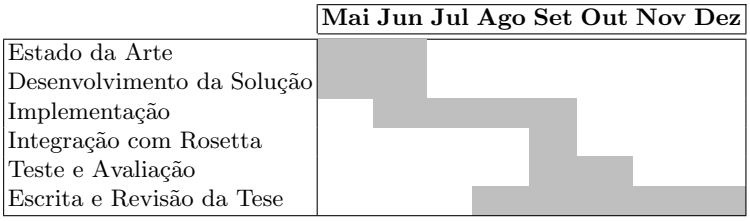
Tabela 2: Lista de ferramentas e bibliotecas para o desenho generativo que permitem fazer restrições geométricas.

Ferramentas	Interface	Características
Cabri[14]	IG	3D; Lógica de construção geométrica clássica (régua e compasso)
CaRMetal[15]	IG	2.5D; Macros; Scripting em JavaScript
Cinderella[16]	IG	Geometria hiperbólica; Scripting em CindyScript
CGAL [14]	LP	Biblioteca de C++ de algoritmos geométricos especializada em robustez
Dr. Geo II[17]	IG	Logica do Smalltalk; Fácil de estender a ferramenta
Eukleides [4]	LP	Linguagem de programação declarativa
GCLC/Wingclc[18]	IG	Construções geométricas basiadas em expressões matemáticas
Geogebra[19]	IG	3D; Expressões matemáticas; CAS; Scripting
GeoSolver [18]	LP	Biblioteca de Python especializada em restrições geométricas
Kaleidoscope [20]	LP	Linguagem de programação de restrição
ThingLab [6]	IG e LP	Paradigma de programação dataflow
TikZ [5]	LP	Desenho do vetorial através especificações

IG — Interface Gráfica
LP — Linguagem de Programação

A.2 Calendarização

Tabela 3: Agendamento de trabalho.



[14] http://www.cabri.com/
[15] http://en.wikipedia.org/wiki/CaRMetal/
[16] http://www.cinderella.de/
[17] http://www.drgeo.eu/
[18] http://poincare.matf.bg.ac.rs/~janicic//gclc/
[19] http://www.geogebra.org/