

# Computer Science in Architecture

António Menezes Leitão

[antonio.menezes.leitao@ist.utl.pt](mailto:antonio.menezes.leitao@ist.utl.pt)

December, 2011

# Preamble

*Ἐν οἴδα ὅτι οὐδὲν οἶδα*

*Σωκράτης*

# Preamble

*Ἐν οἴδα ὅτι οὐδὲν οἶδα*

*Σωκράτης*

*I know one thing, that I know nothing*

*Socrates*

# Architecture and Computer Science

*Et ut litteratus sit, peritus graphidos, eruditus  
geometria, historias complures noverit, musicam  
scierit, medicinae non sit ignarus*

*Marcus Vitruvius Pollio*

# Architecture and Computer Science

*Et ut litteratus sit, peritus graphidos, eruditus  
geometria, historias complures noverit, musicam  
scierit, medicinae non sit ignarus*

*Marcus Vitruvius Pollio*

An architect must be:

- a good writer

# Architecture and Computer Science

*Et ut litteratus sit, peritus graphidos, eruditus  
geometria, historias complures noverit, musicam  
scierit, medicinae non sit ignarus*

*Marcus Vitruvius Pollio*

An architect must be:

- a good writer
- a skilful draftsman

# Architecture and Computer Science

*Et ut litteratus sit, peritus graphidos, eruditus  
geometria, historias complures noverit, musicam  
scierit, medicinae non sit ignarus*

*Marcus Vitruvius Pollio*

An architect must be:

- a good writer
- a skilful draftsman
- versed in geometry

# Architecture and Computer Science

*Et ut litteratus sit, peritus graphidos, eruditus  
geometria, historias complures noverit, musicam  
scierit, medicinae non sit ignarus*

*Marcus Vitruvius Pollio*

An architect must be:

- a good writer
- a skilful draftsman
- versed in geometry
- acquainted with history

# Architecture and Computer Science

*Et ut litteratus sit, peritus graphidos, eruditus  
geometria, historias complures noverit, musicam  
scierit, medicinae non sit ignarus*

*Marcus Vitruvius Pollio*

An architect must be:

- a good writer
- a skilful draftsman
- versed in geometry
- acquainted with history
- a musician

# Architecture and Computer Science

*Et ut litteratus sit, peritus graphidos, eruditus  
geometria, historias complures noverit, musicam  
scierit, medicinae non sit ignarus*

*Marcus Vitruvius Pollio*

An architect must be:

- a good writer
- a skilful draftsman
- versed in geometry
- acquainted with history
- a musician
- not ignorant of the sciences

# Architecture and Computer Science

*Et ut litteratus sit, peritus graphidos, eruditus  
geometria, historias complures noverit, musicam  
scierit, medicinae non sit ignarus, eruditus computare*

*Marcus Vitruvius Pollio*

An architect must be:

- a good writer
- a skilful draftsman
- versed in geometry
- acquainted with history
- a musician
- not ignorant of the sciences
- versed in computation



# Architecture needs Computer Science

# Architecture needs Computer Science

How do we teach Computer Science to Architects?

# Architecture needs Computer Science

How do we teach Computer Science to Architects?

- Geometry

# Architecture needs Computer Science

How do we teach Computer Science to Architects?

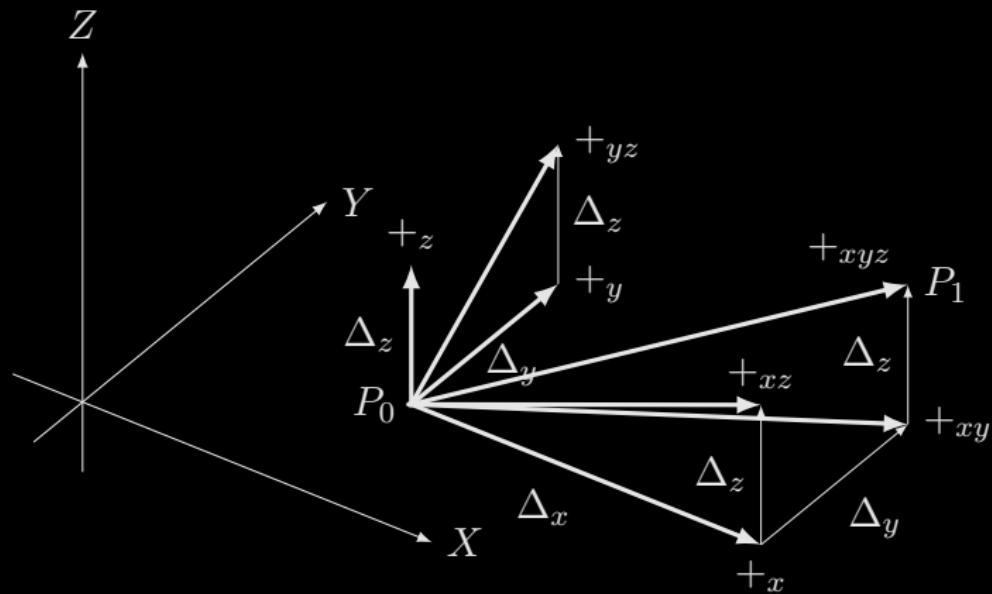
- Geometry
- Algorithms

# Architecture needs Computer Science

How do we teach Computer Science to Architects?

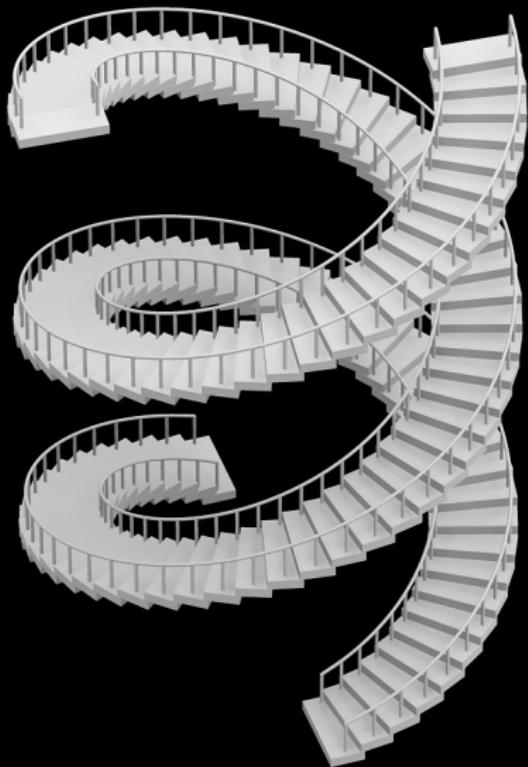
- Geometry
- Algorithms
- Programming Language

# Geometry - Rectangular Coordinates

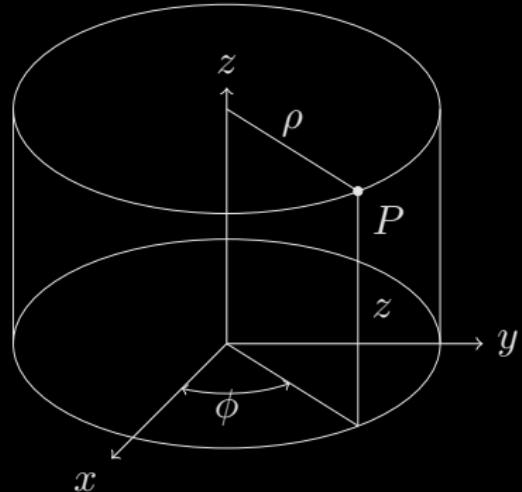
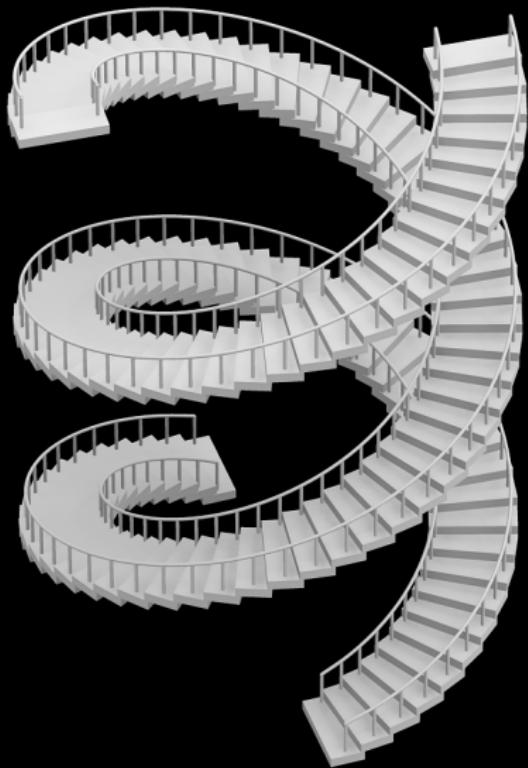


$$P_1 = +xyz(P_0, \Delta_x, \Delta_y, \Delta_z)$$

# Geometry - Cylindrical Coordinates



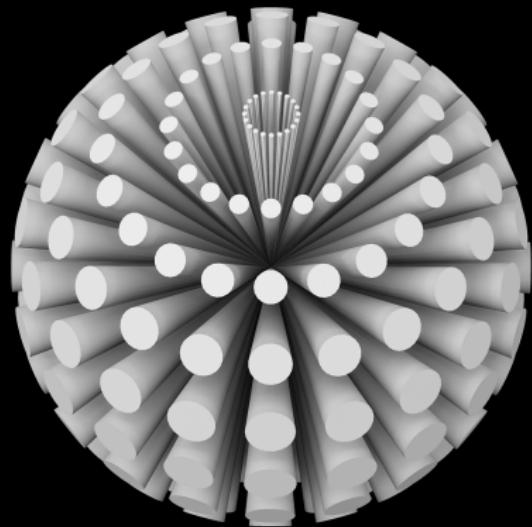
# Geometry - Cylindrical Coordinates



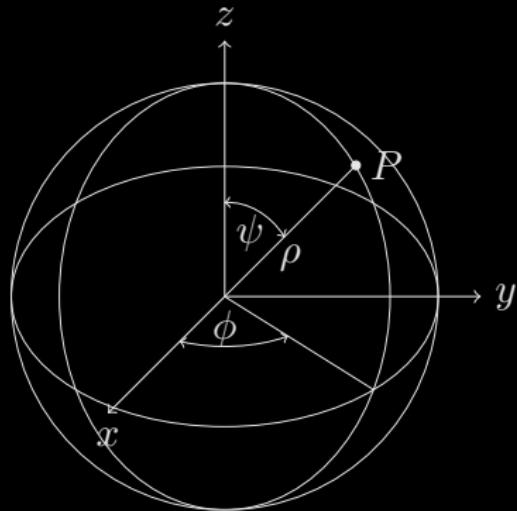
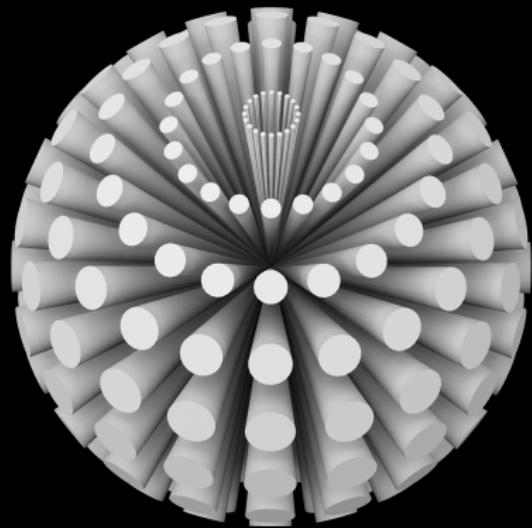
$$\forall s \in [0, \dots, n]$$

**step( cyl( $r_i, \Delta_\omega \cdot s, \Delta_h \cdot s$ ),  
cyl( $r_e, \Delta_\omega \cdot s, \Delta_h \cdot s$ ) )**

# Geometry - Spherical Coordinates



# Geometry - Spherical Coordinates

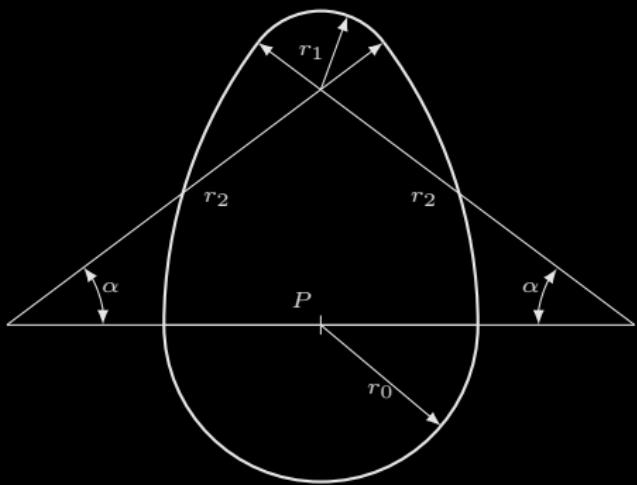


$$\forall_{\psi} \in \{0, \frac{\pi}{10}, \dots, \pi\}$$

$$\forall_{\phi} \in \{\frac{\pi}{10}, 2\frac{\pi}{10}, \dots, 2\pi\}$$

`cone(sph(1, φ, ψ), sin ψ / 10, xyz(0, 0, 0))`

# Geometry - Primitives



# Geometry - Primitives

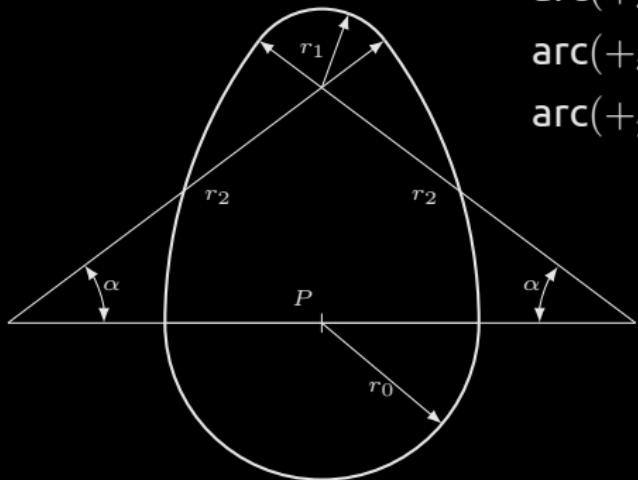
**egg**( $p, r_0, r_1, h) =$

**arc**( $p, r_0, 0, -\pi) \cup$

**arc**( $+_x(p, r_0 - r_2), r_2, 0, \alpha) \cup$

**arc**( $+_x(p, r_2 - r_0), r_2, \alpha - \pi, \alpha) \cup$

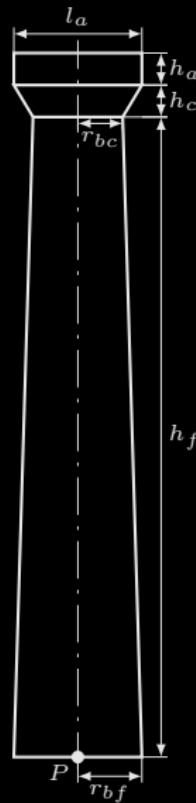
**arc**( $+_y(p, (r_2 - r_1) \sin \alpha), r_1, \alpha, \pi - 2\alpha)$



$$\alpha = 2 \tan^{-1} \frac{r_0 - r_1}{h - r_0 - r_1}$$

$$r_2 = \frac{r_0 - r_1 \cos \alpha}{1 - \cos \alpha}$$

# Algorithms



# Algorithms



```
column( $p, h_f, r_{bf}, h_c, r_{bc}, h_a, l_a$ ) =  
    frustum( $p, h_f, r_{bf}, r_{bc}$ )  $\cup$   
    frustum( $+z(p, h_f), h_c, r_{bc}, l_a/2$ )  $\cup$   
    box( $+z(p, h_f + h_c), h_a, l_a$ )
```

# Algorithms



*Crassitudo columnarum erit  
duorum modulorum, altitudo cum  
capitulo XIII. Capituli crassitudo  
unius moduli, latitudo duorum et  
moduli sextae partis. Crassitudo  
capituli dividatur in partes tres, e  
quibus una plinthus cum cymatio  
fiat, altera echinus cum anulis,  
tertia hypotrachelion.*

Vitruvius

# Algorithms



$$h_f = 13r_{bf}$$

$$h_c = \frac{2}{3}r_{bf}$$

$$h_a = \frac{1}{3}r_{bf}$$

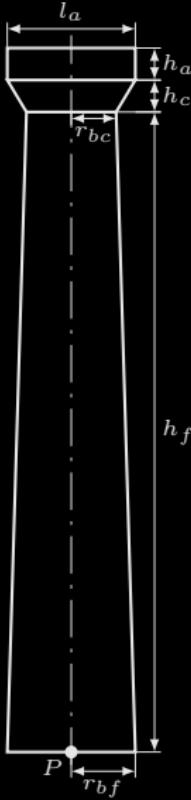
$$l_a = \frac{13}{6}r_{bf}$$

# Algorithms



**column**( $p, h_f, r_{bf}, h_c, r_{bc}, h_a, l_a$ ) =  
**frustrum**( $p, h_f, r_{bf}, r_{bc}$ )  $\cup$   
**frustrum**( $+_z(p, h_f), h_c, r_{bc}, l_a/2$ )  $\cup$   
**box**( $+_z(p, h_f + h_c), h_a, l_a$ )

# Algorithms



**column**( $p, h_f, r_{bf}, r_{bc}, h_a, l_a$ ) =  
**frustrum**( $p, h_f, r_{bf}, r_{bc}$ )  $\cup$   
**frustrum**( $+_z(p, h_f), h_c, r_{bc}, l_a/2$ )  $\cup$   
**box**( $+_z(p, h_f + h_c), h_a, l_a$ )

**doricColumn**( $p, r_{bf}, r_{bc}$ ) =  
**column**( $p, \frac{2}{3}r_{bf}, r_{bf}, \frac{1}{3}r_{bf}, r_{bc}, \frac{13}{6}r_{bf}$ )

# Programming Languages

# Programming Languages



A word cloud visualization where the size of each programming language name corresponds to its frequency or popularity. The most prominent names include Fortran, Java, C, Python, and C++, while many others like Brainfuck, LeLisp, and VBA are much smaller.

Fortran Javascript Forth Ruby Basic VisualBasic Java Grasshopper Smalltalk Scheme Pascal MP4 C+ TeX Common-Lisp Objective-C Rhinoscript Groovy Bash Prolog Brainfuck ZetaLisp SQL Lisp Datalog FranzLisp Eiffel VBA

# Programming Languages

Most languages have the same *computational power*

# Programming Languages

Most languages have the same *computational power*

but they differ in

- clarity
- learning curve
- paradigms
- execution speed
- productivity
- ...
- *expressive power*

# Programming Languages - Clarity

## Visual Basic

```
Function ConicSpiralPoints(Length,N)
    Dim points()
    ReDim points(N-1)
    Dim t, i
    For i=0 To N-1
        t=i*Length/N
        points(i)=Pt(t*Cos(5*t),t*Sin(5*t),t)
    Next
    ConicSpiralPoints=points
End Function
```

# Programming Languages - Clarity

## Haskell

```
conicSpiralPoints length n =  
[(t*(cos 5*t), t*(sin 5*t), t) |  
 t <- [i*length/n | i <- [0..n]]]
```

# Programming Languages - Choices

- It must be simple
- It must be expressive
- It must be usable with a CAD application

# Programming Languages - Choices

- It must be simple
- It must be expressive
- It must be usable with a CAD application
  
- ArchiCad
  - GDL
- AutoCAD
  - Visual Basic
  - C++
  - AutoLisp
- Rhino
  - RhinoScript
  - Grasshopper
  - Python

# Programming Languages - My Choice

*Lisp isn't a language, it's a building material.*

*Alan Kay*

# Programming Languages - My Choice

*Lisp isn't a language, it's a building material.*

*Alan Kay*

*Lisp made me aware that software could be close to executable mathematics.*

*Peter Deutsch*

# Programming Languages - My Choice

*Lisp isn't a language, it's a building material.*

*Alan Kay*

*Lisp made me aware that software could be close to executable mathematics.*

*Peter Deutsch*

*Lisp is worth learning for the profound enlightenment experience you will have when you finally get it; that experience will make you a better programmer for the rest of your days, even if you never actually use Lisp itself a lot*

*Eric S. Raymond*

# Course Outline

# Course Outline

- Syntax & Semantics
- Recursion
- State & Randomness
- Lists
- Solid Geometry
- Higher Order Functions
- Parametrics

# Syntax & Semantics

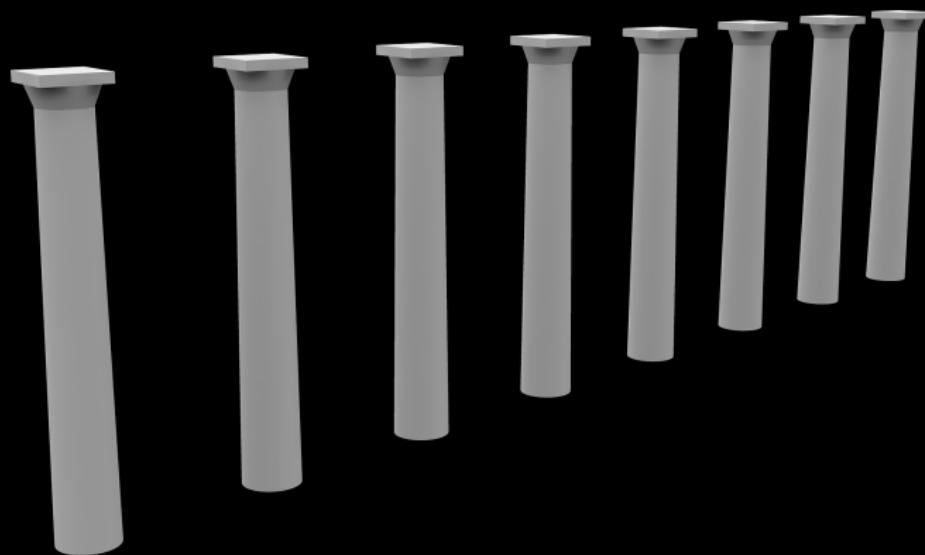
```
(defun cross (p rb rt c)
  (cone-frustrum
    p rb (+x p c) rt)
  (cone-frustrum
    p rb (+y p c) rt)
  (cone-frustrum
    p rb (+z p c) rt)
  (cone-frustrum
    p rb (+x p (- c)) rt)
  (cone-frustrum
    p rb (+y p (- c)) rt)
  (cone-frustrum
    p rb (+z p (- c)) rt))
```

# Recursion

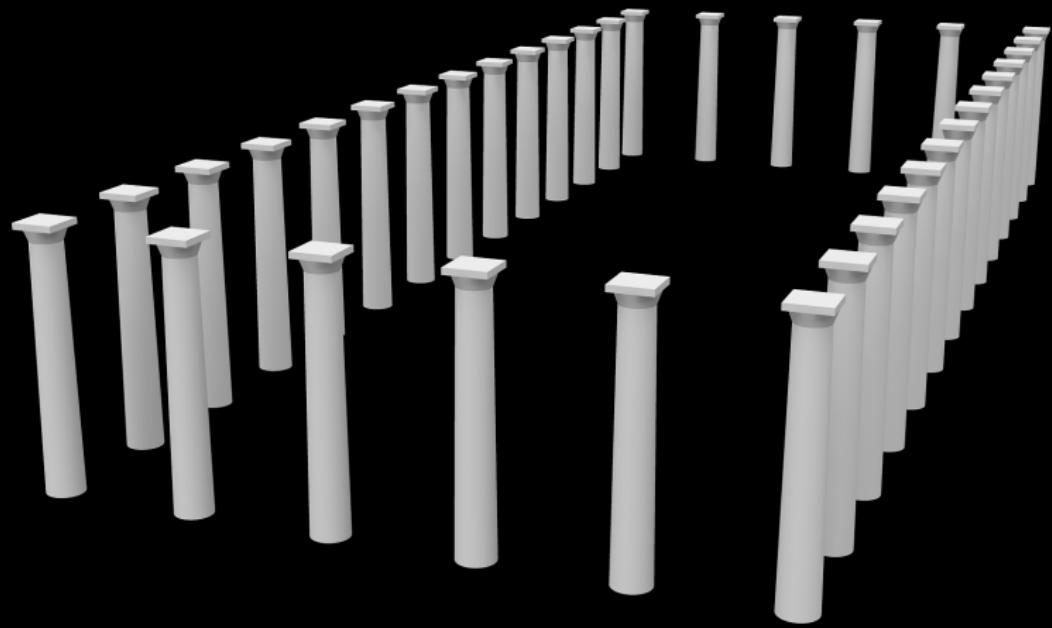
```
(defun colonnade (p h v n)
  (if (= n 0)
      nil
      (progn
        (column p h)
        (colonnade (+c p v) h v (- n 1)))))
```

# Recursion

```
(defun colonnade (p h v n)
  (if (= n 0)
      nil
      (progn
        (column p h)
        (colonnade (+c p v) h v (- n 1)))))
```



# Recursion



# Recursion

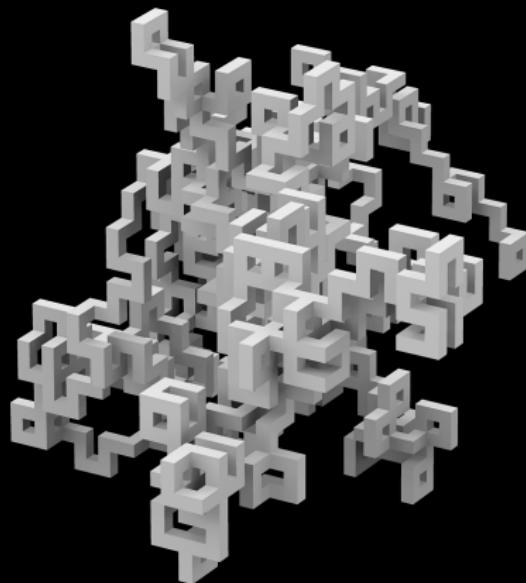
```
(defun menger (p l n)
  (if (= n 0)
      (cube p 1)
      (progn
        (setq l (/ l 3.0))
        (foreach i (enum 0 2 1)
          (foreach j (enum 0 2 1)
            (foreach k (enum 0 2 1)
              (if (or (= i j 1) (= i k 1) (= j k 1))
                  nil
                  (menger
                    (+xyz p (* i 1) (* j 1) (* k 1))
                    l
                    (- n 1))))))))))
```

# State & Randomness

```
(defun random-direction ()  
  (sph 1  
    (* (random-[] 0 4) pi/2)  
    (* (random-[] 0 4) pi/2)))
```

# State & Randomness

```
(defun random-direction ()  
  (sph 1  
    (* (random-[] 0 4) pi/2)  
    (* (random-[] 0 4) pi/2)))
```



# State & Randomness

# State & Randomness

## Gaussian Distribution

$$f(x, y) = e^{-\left(\left(\frac{x-x_0}{\sigma_x}\right)^2 + \left(\frac{y-y_0}{\sigma_y}\right)^2\right)}$$

# Lists

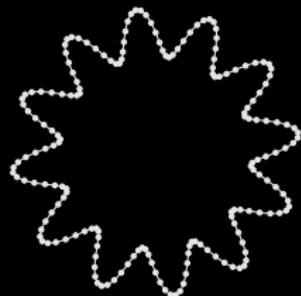
# Lists

## Curves from Points



# Lists

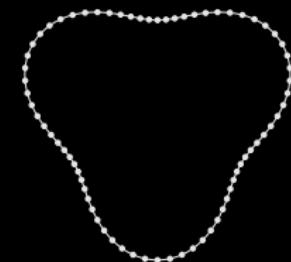
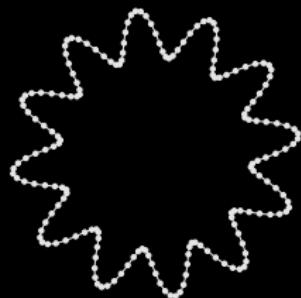
## Curves from Points



```
(defun sinusoidal-arc-points (p r a c fi dfi n)
  (if (= n 0)
      (list)
      (cons (+pol p (+ r (* a (sin (* c fi))))) fi)
            (sinusoidal-arc-points p r a c (+ fi dfi) dfi (- n 1))))
```

# Lists

## Curves from Points

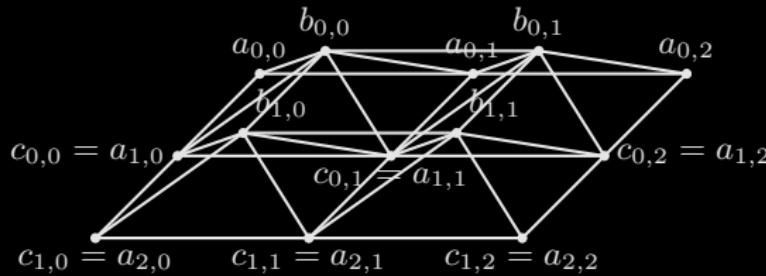


```
(defun sinusoidal-arc-points (p r a c fi dfi n)
  (if (= n 0)
      (list)
      (cons (+pol p (+ r (* a (sin (* c fi))))) fi)
            (sinusoidal-arc-points p r a c (+ fi dfi) dfi (- n 1)))))

(defun sinusoide-circular-arc (p ri re c n)
  (sinusoidal-arc-points
    p (/ (+ ri re) 2.0) (/ (- re ri) 2.0)
    c 0 (/ 2*pi n) n))
```

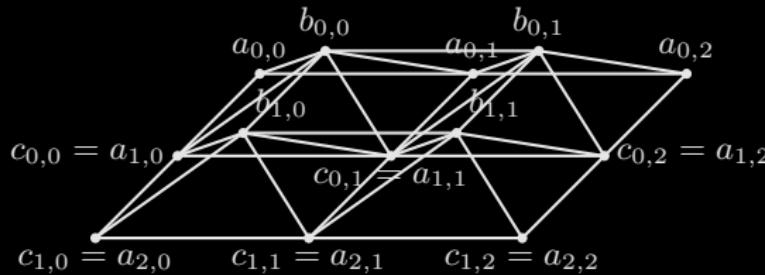
# Lists

## Trusses



# Lists

## Trusses



$$((a_{0,0} \quad a_{0,1} \quad a_{0,2} \quad \dots \quad a_{0,n-1} \quad a_{0,n})$$

$$(b_{0,0} \quad b_{0,1} \quad b_{0,2} \quad \dots \quad b_{0,n-1})$$

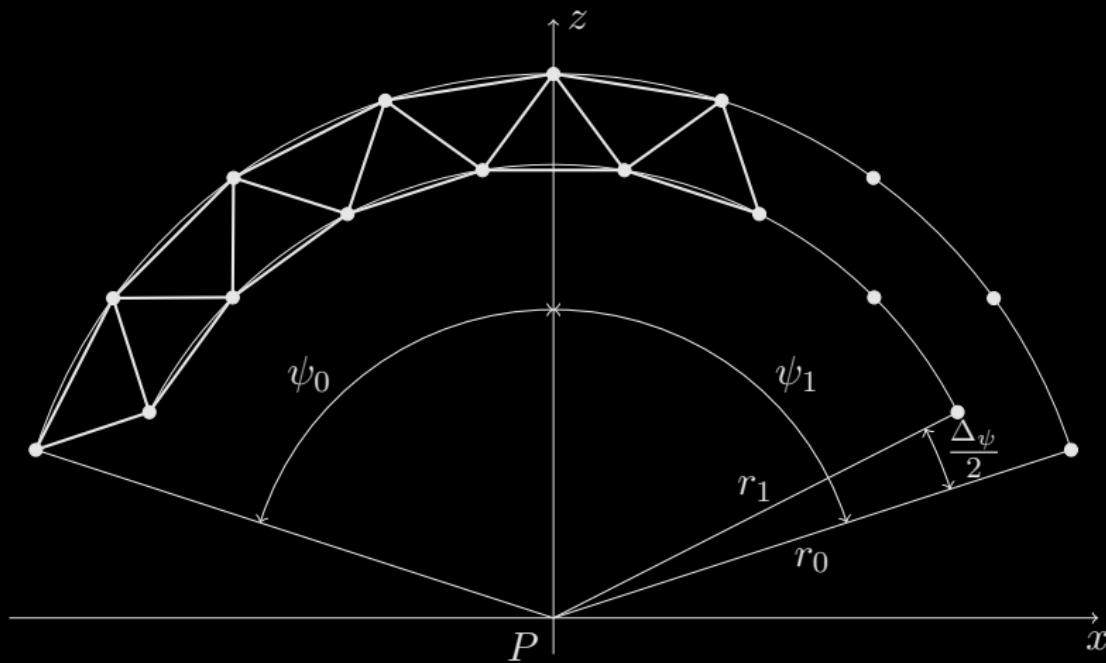
$$(a_{1,0} \quad a_{1,1} \quad a_{1,2} \quad \dots \quad a_{1,n-1} \quad a_{1,n})$$

...

$$(c_{m,0} \quad c_{m,1} \quad c_{m,2} \quad \dots \quad c_{m,n-1} \quad c_{m,n}))$$

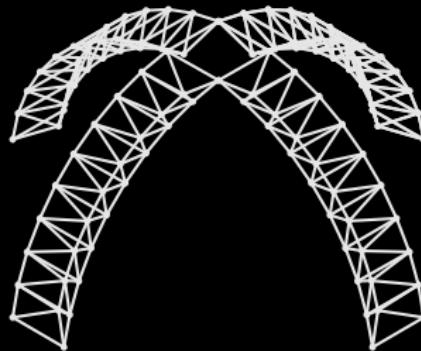
# Lists

## Trusses



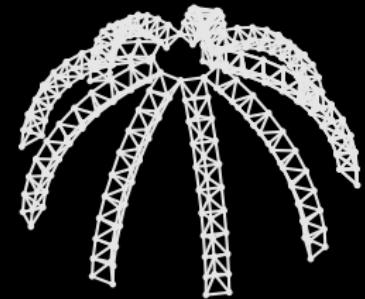
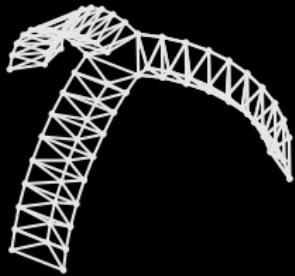
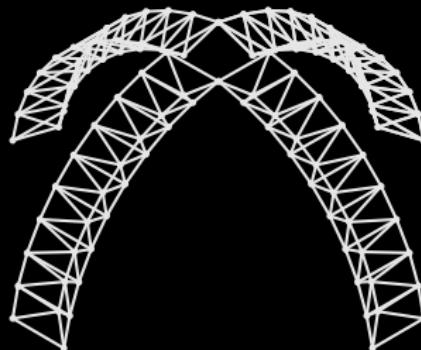
# Lists

## Trusses



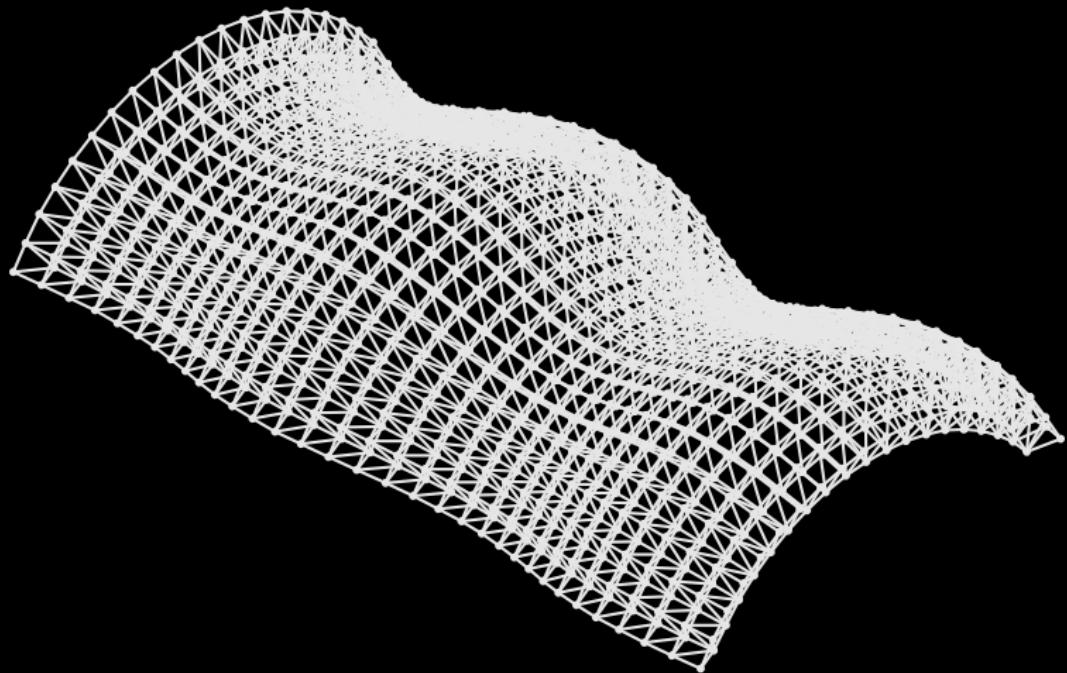
# Lists

## Trusses



# Lists

## Trusses



# Solid Geometry

## Region Algebra

$$R \cup \emptyset = \emptyset \cup R = R$$

$$R \cup U = U \cup R = U$$

$$R \cup R = R$$

$$R \cap \emptyset = \emptyset \cap R = \emptyset$$

$$R \cap U = U \cap R = R$$

$$R \cap R = R$$

$$R \setminus \emptyset = R$$

# Solid Geometry

## Region Algebra

$$R \cup \emptyset = \emptyset \cup R = R$$

$$R \cup U = U \cup R = U$$

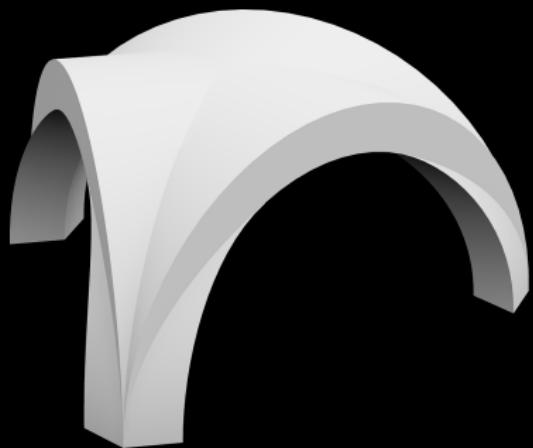
$$R \cup R = R$$

$$R \cap \emptyset = \emptyset \cap R = \emptyset$$

$$R \cap U = U \cap R = R$$

$$R \cap R = R$$

$$R \setminus \emptyset = R$$



# Solid Geometry

# Solid Geometry

# Solid Geometry



# Solid Geometry

## Operations

- Regular polygon
- Region
- Sweep
- Rotation
- Scale
- Intersection

# Solid Geometry



# Solid Geometry

## Operations

- Sphere
- Subtraction
- Section
- Translation
- Reflection
- Duplication
- Rotation
- Scale



# Solid Geometry

## Mathematical Functions

$$y(x) = a \sin(\omega x + \phi)$$

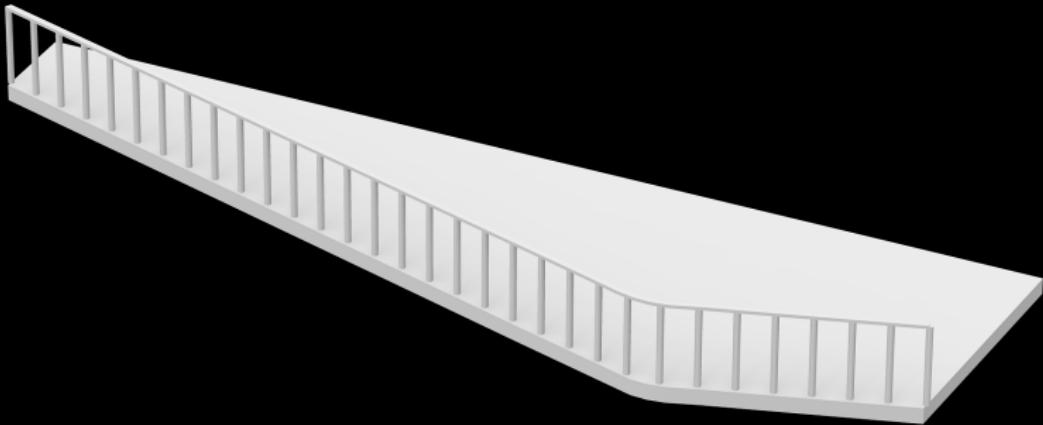
# Solid Geometry

## Possible Variations

```
(defun building (...)  
  ...  
  (balcony ...)  
  ...)  
  
(defun balcony (...)  
  ... (* a (sin (+ (* omega x) phi))) ...)
```

# Solid Geometry

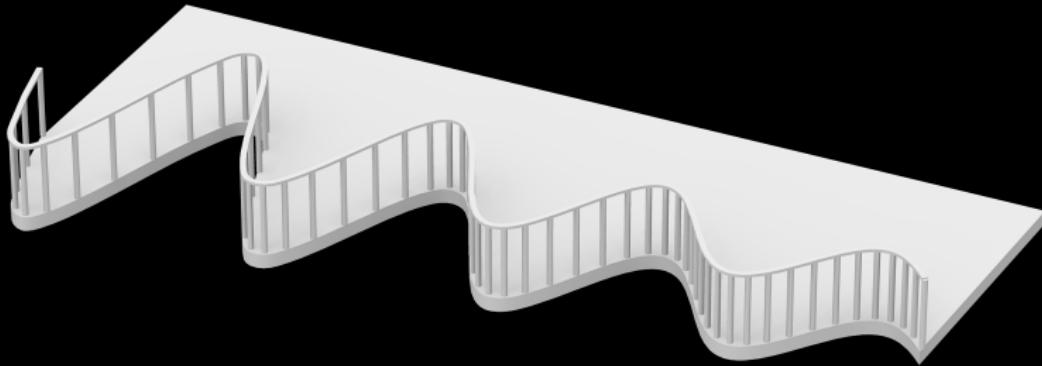
## Impossible Variations



$$y(x) = |a + b(x - c)|$$

# Solid Geometry

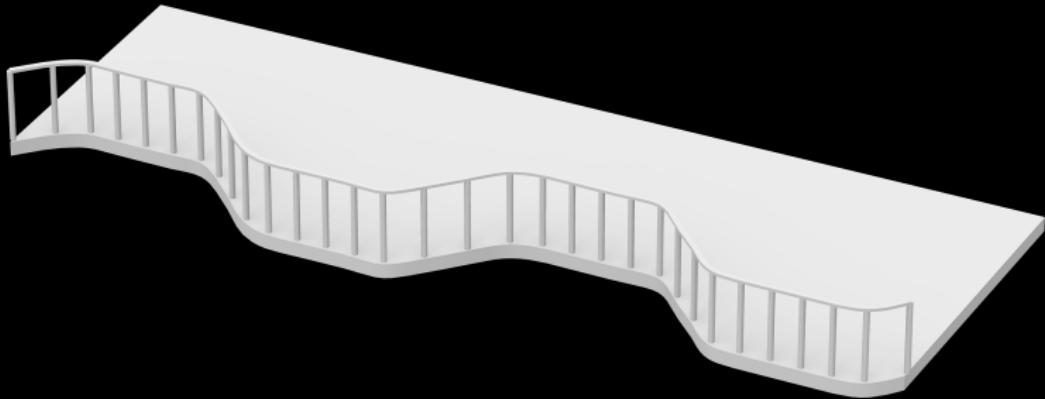
## Impossible Variations



$$y(x) = ae^{-bx} \sin(cx)$$

# Solid Geometry

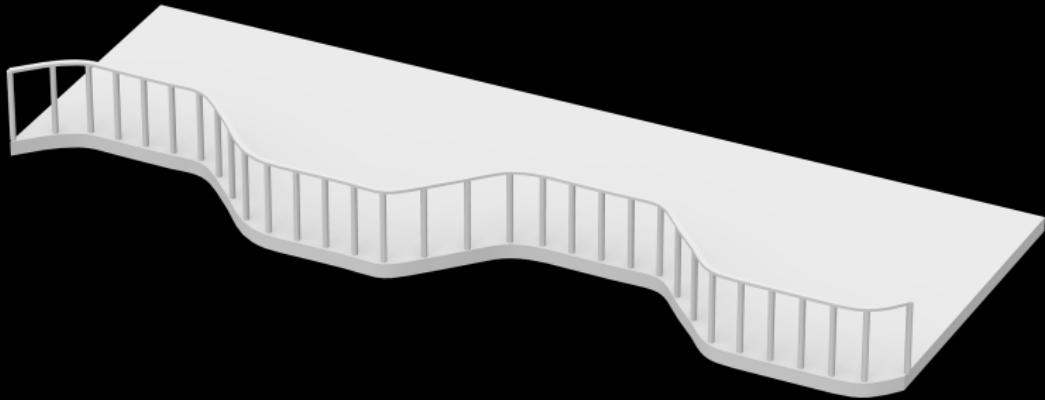
## Impossible Variations



$$y(x) = \max(y_0, \min(y_1, a \sin(\omega x + \phi)))$$

# Solid Geometry

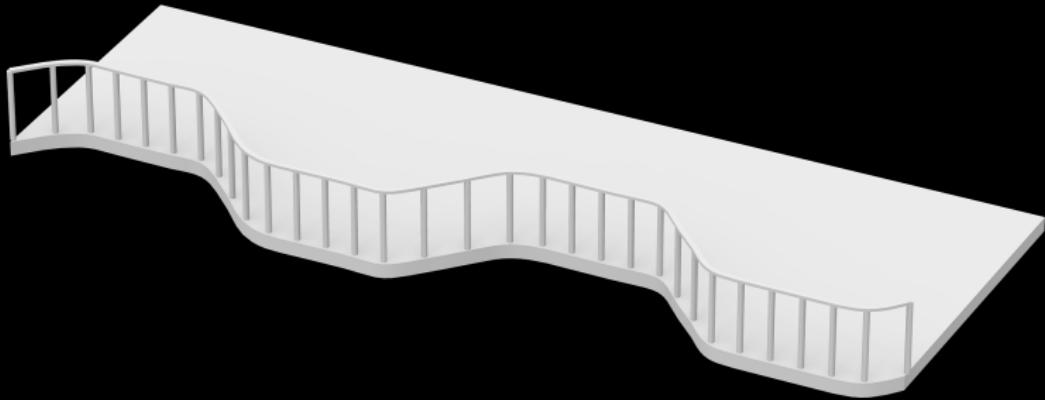
## Impossible Variations



```
(defun building (...)  
  ...  
  (balcony ...) ...)
```

# Solid Geometry

## Impossible Variations



```
(defun building (... balcony ...)  
  ...  
  (balcony ...)  
  ...)
```

# Higher Order Functions

## Definition

A function that receives or returns functions

# Higher Order Functions

## Definition

A function that receives or returns functions

## Example

$$\sum_{i=m}^n f(i) = \begin{cases} 0, & \text{if } m > n \\ x_m + \sum_{i=m+1}^n f(i), & \text{otherwise.} \end{cases}$$

# Higher Order Functions

## Definition

A function that receives or returns functions

## Example

$$\sum_{i=m}^n f(i) = \begin{cases} 0, & \text{if } m > n \\ x_m + \sum_{i=m+1}^n f(i), & \text{otherwise.} \end{cases}$$

$$\frac{d}{dx} f(x) = f'(x) = D_x f(x) = \lim_{\Delta_x \rightarrow 0} \frac{f(x + \Delta_x) - f(x)}{\Delta_x}$$

# Higher Order Functions

## Definition

A function that receives or returns functions

## Example

$$\sum_{i=m}^n f(i) = \begin{cases} 0, & \text{if } m > n \\ x_m + \sum_{i=m+1}^n f(i), & \text{otherwise.} \end{cases}$$

$$\frac{d}{dx} f(x) = f'(x) = D_x f(x) = \lim_{\Delta_x \rightarrow 0} \frac{f(x + \Delta_x) - f(x)}{\Delta_x}$$

$$f \circ g = \lambda(x) f(g(x))$$

# Parametric Curves

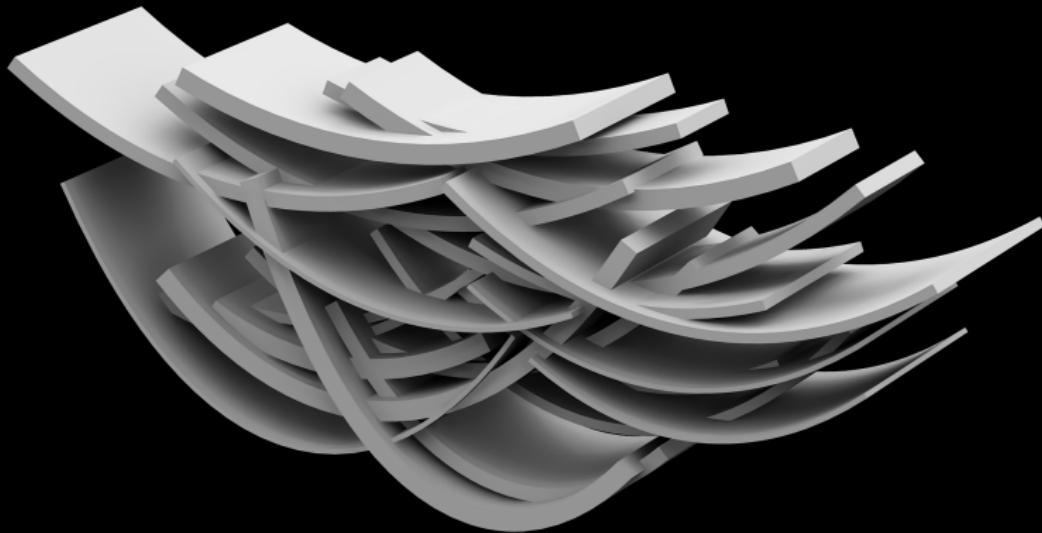
## Catenary

$$y = a \cosh \frac{x}{a}$$

# Parametric Curves

## Catenary

$$y = a \cosh \frac{x}{a}$$



# Parametric Curves

## Catenary

$$y = a \cosh \frac{x}{a}$$



# Parametric Surfaces

Möbius Strip



# Parametric Surfaces

## Möbius Strip



$$\begin{cases} \rho(u, v) = 1 + v \cos \frac{u}{2} \\ \theta(u, v) = u \\ z(u, v) = v \sin \frac{u}{2} \end{cases}$$

# Parametric Surfaces

Möbius Strip



# Parametric Surfaces

## Möbius Strip



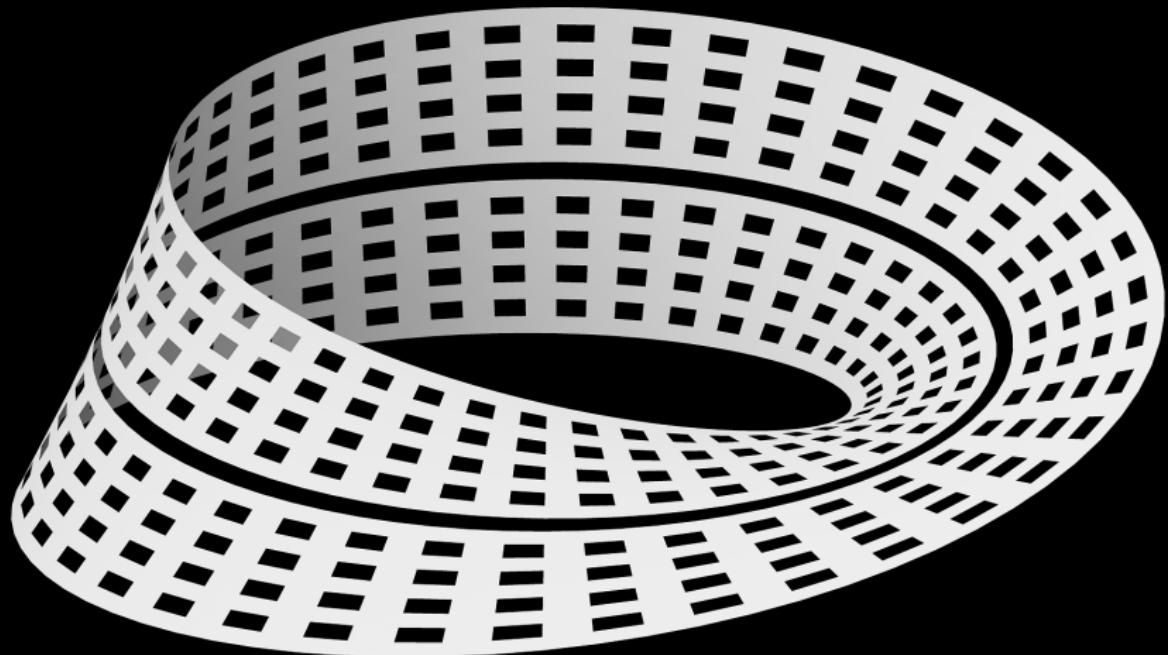
```
(defun moebius-strip (u0 u1 m v0 v1 n)
  (parametric-surface
    (lambda (u v)
      (cyl (+ 1 (* v (cos (/ u 2.0)))))
        u
        (* v (sin (/ u 2.0))))))
  u0 u1 m v0 v1 n))
```

# Parametric Surfaces

## Möbius Strip

# Parametric Surfaces

Möbius Strip

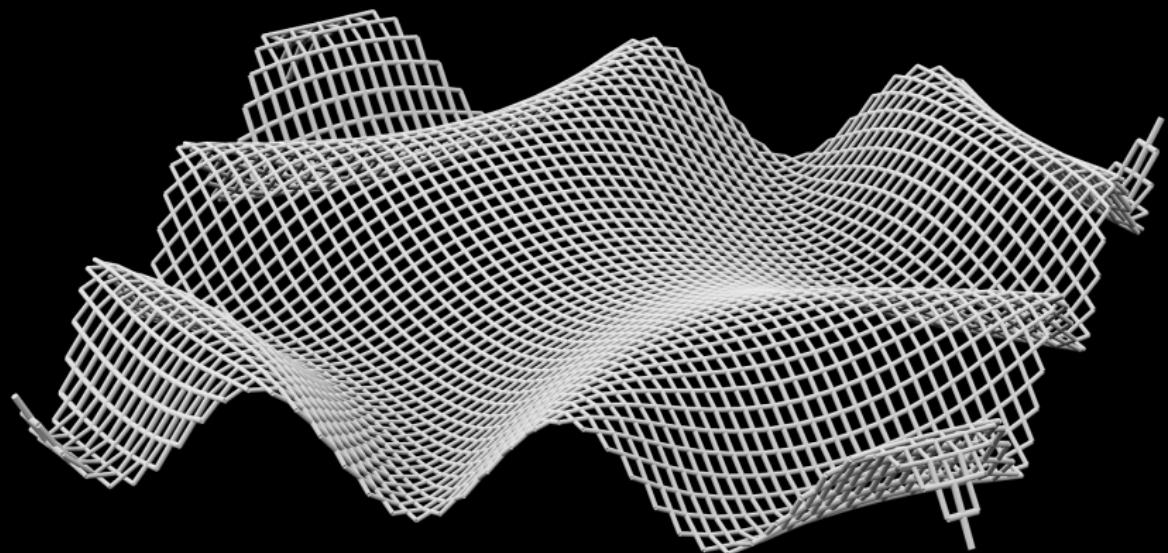


# Parametric Surfaces

## Tesselations

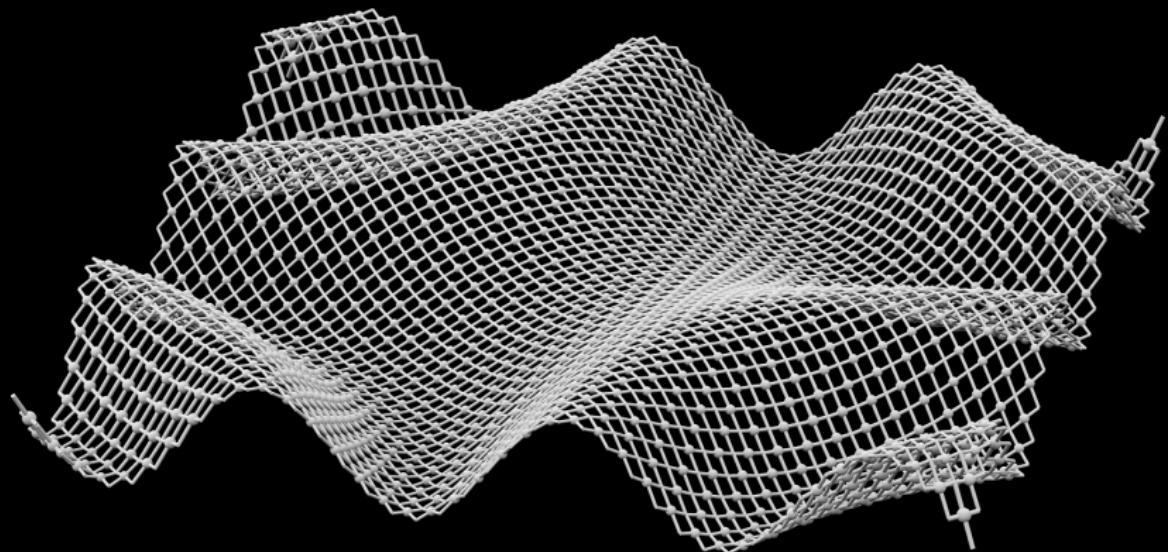
# Parametric Surfaces

## Tesselations



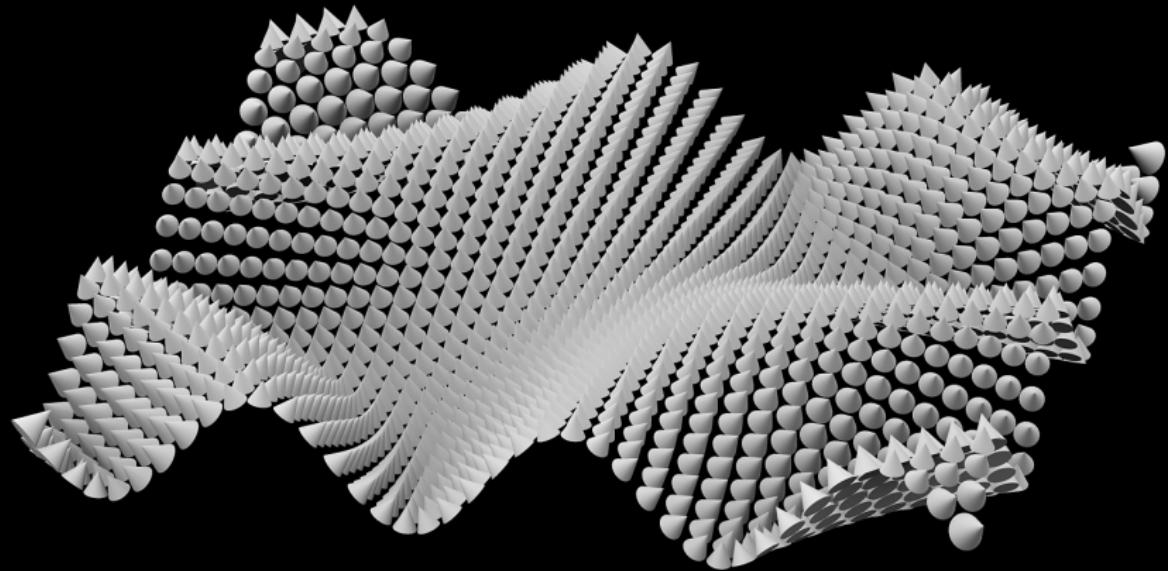
# Parametric Surfaces

## Tesselations



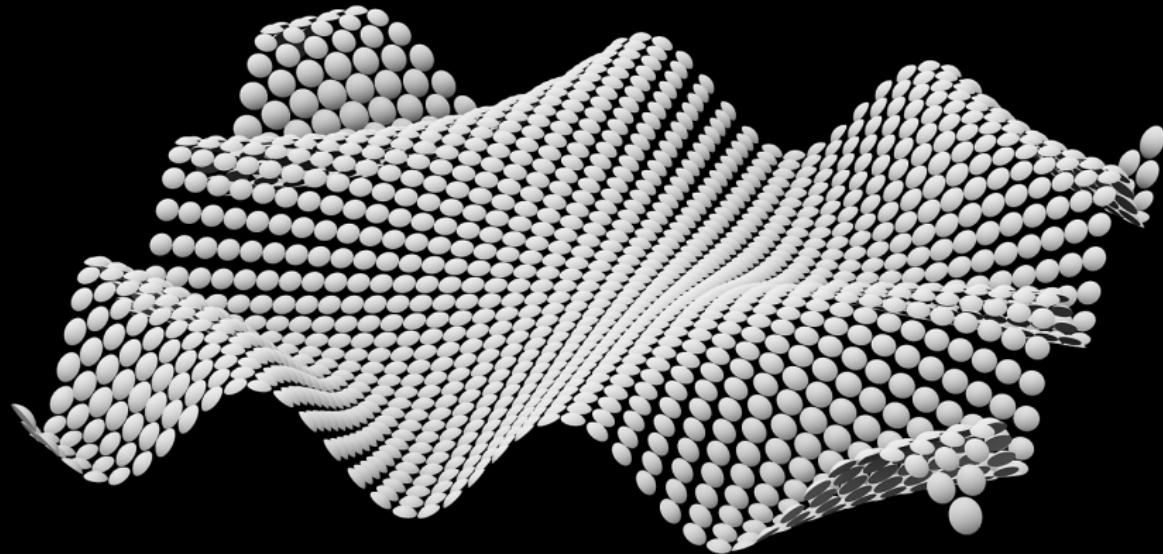
# Parametric Surfaces

## Tesselations



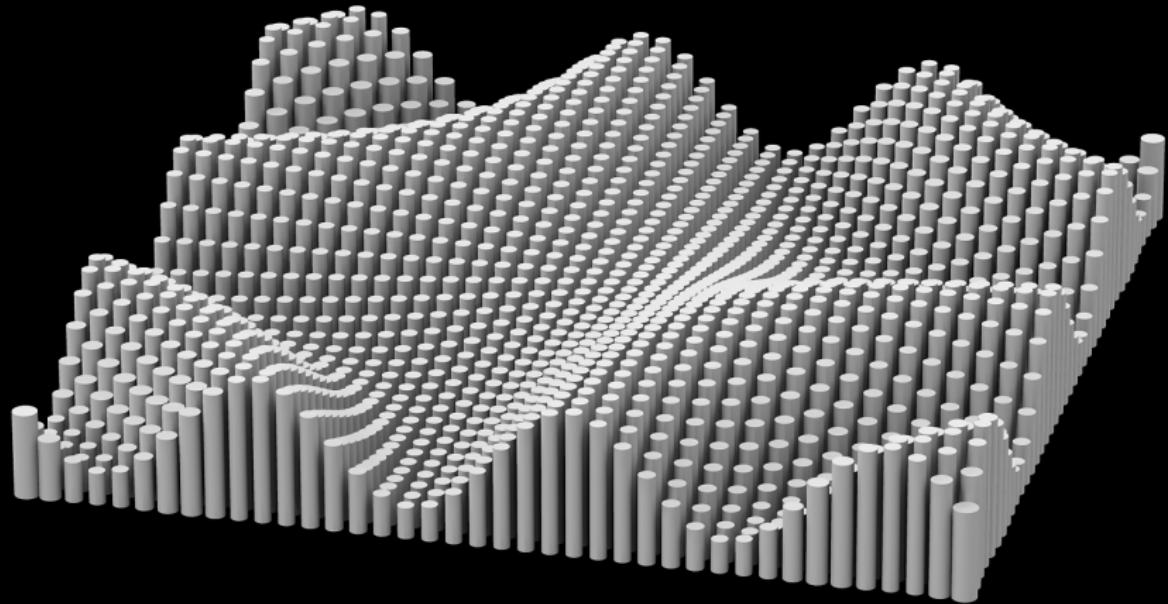
# Parametric Surfaces

## Tesselations



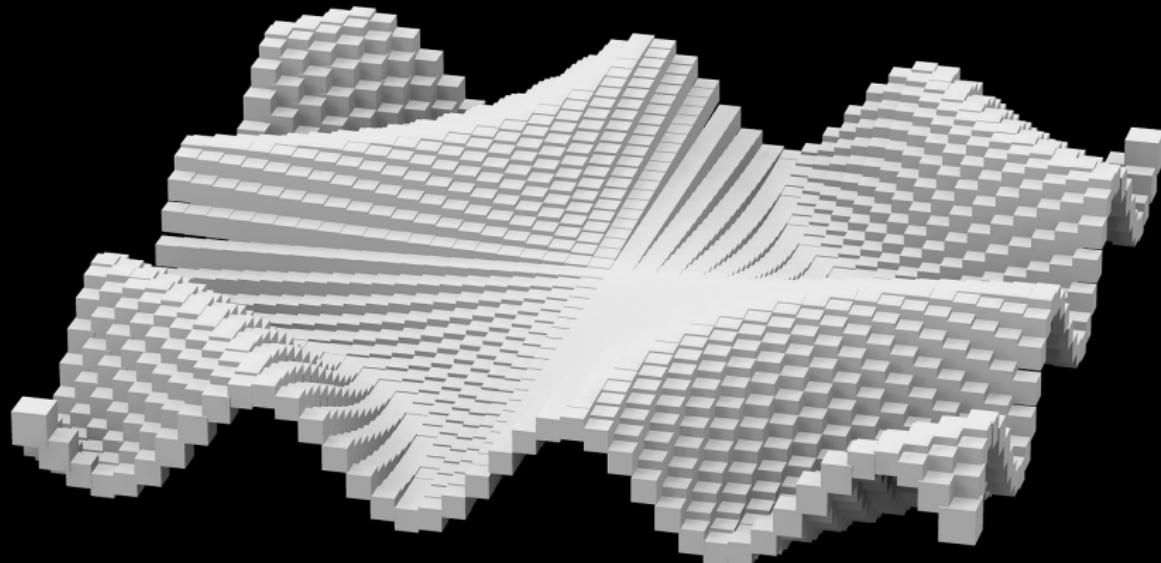
# Parametric Surfaces

## Tesselations



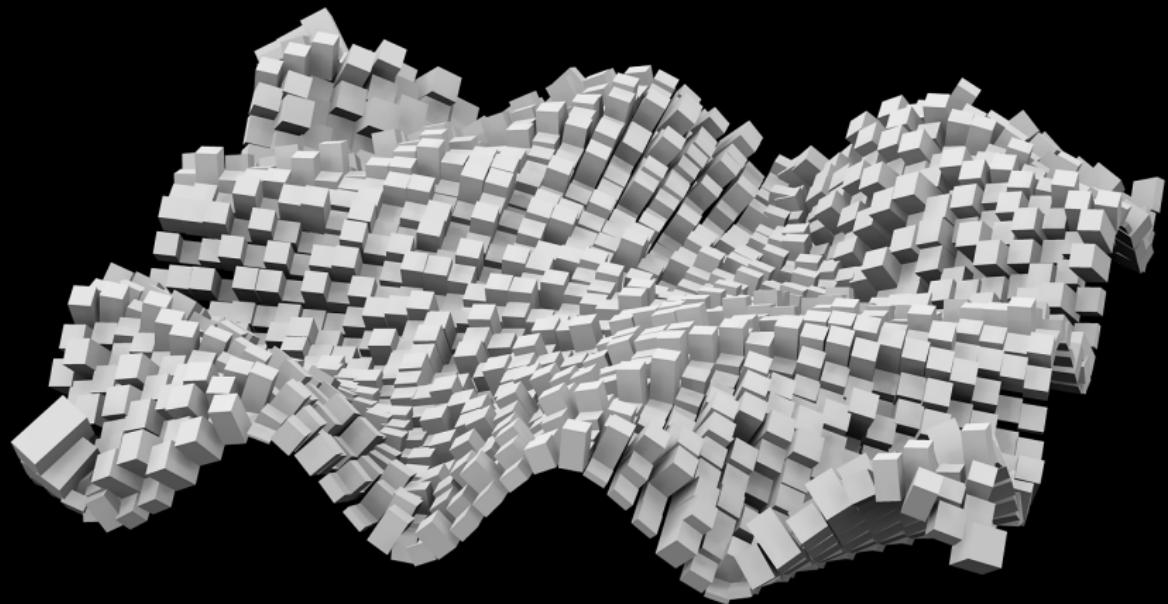
# Parametric Surfaces

## Tesselations



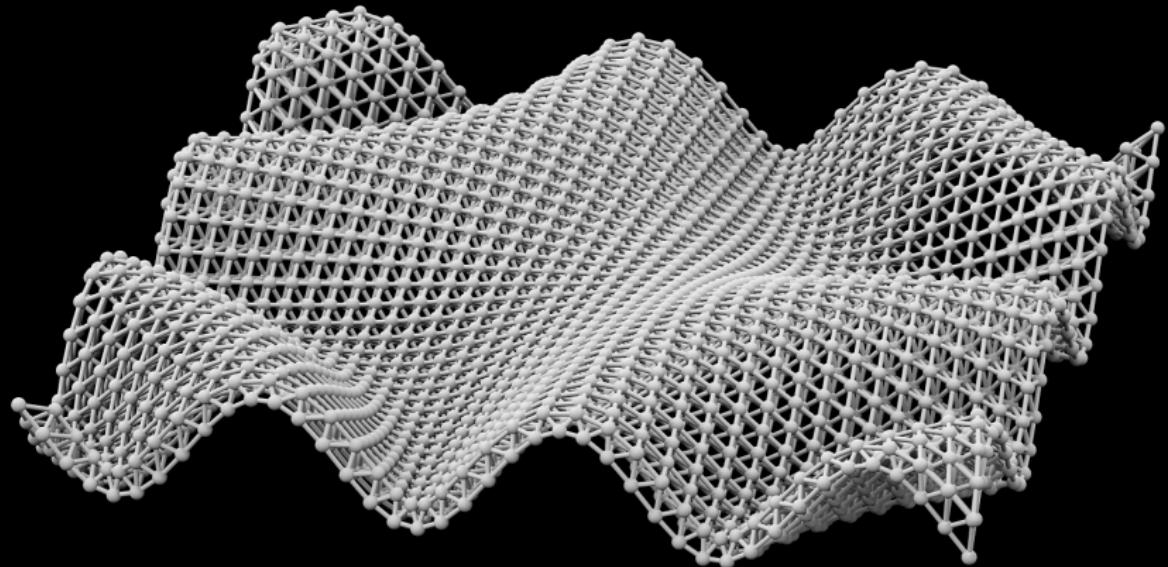
# Parametric Surfaces

## Tesselations



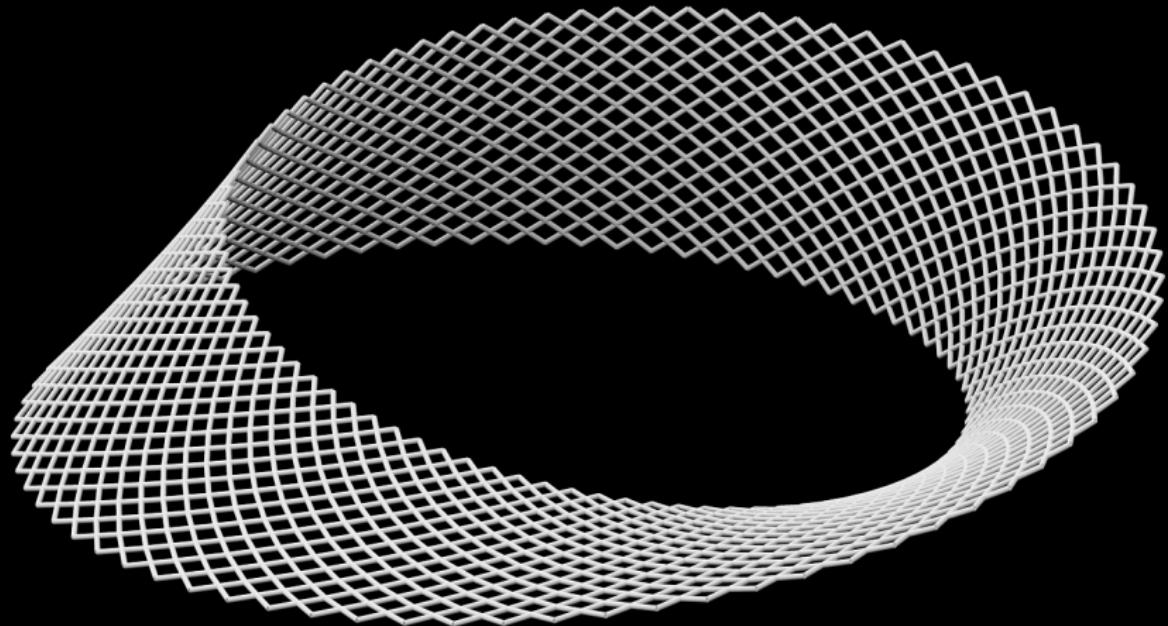
# Parametric Surfaces

## Tesselations



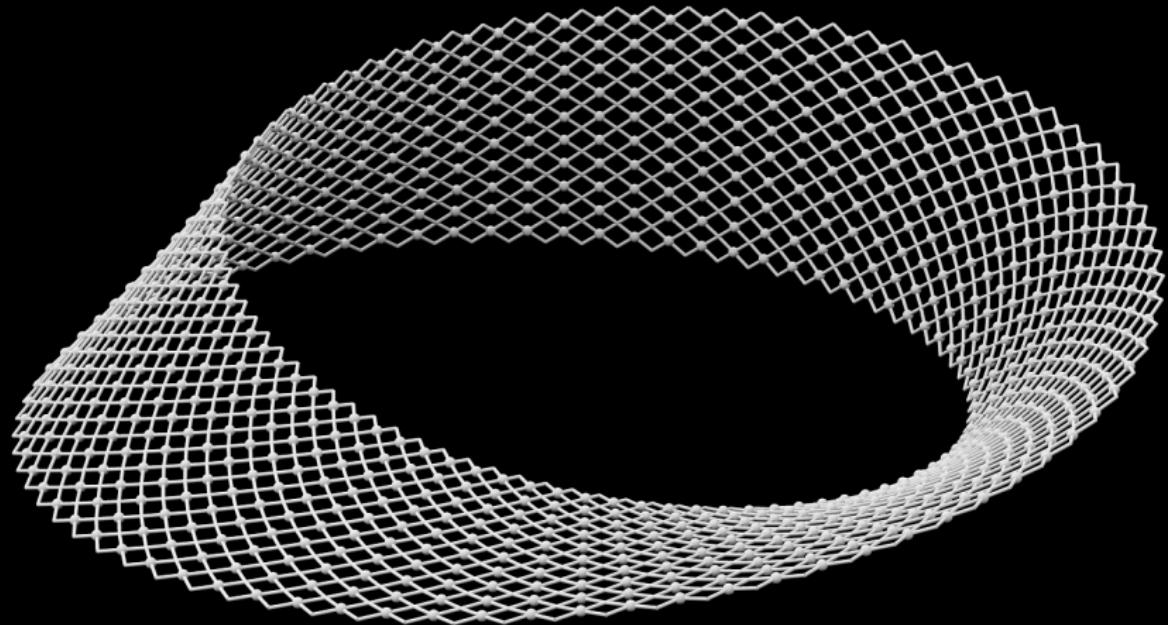
# Parametric Surfaces

## Tesselations



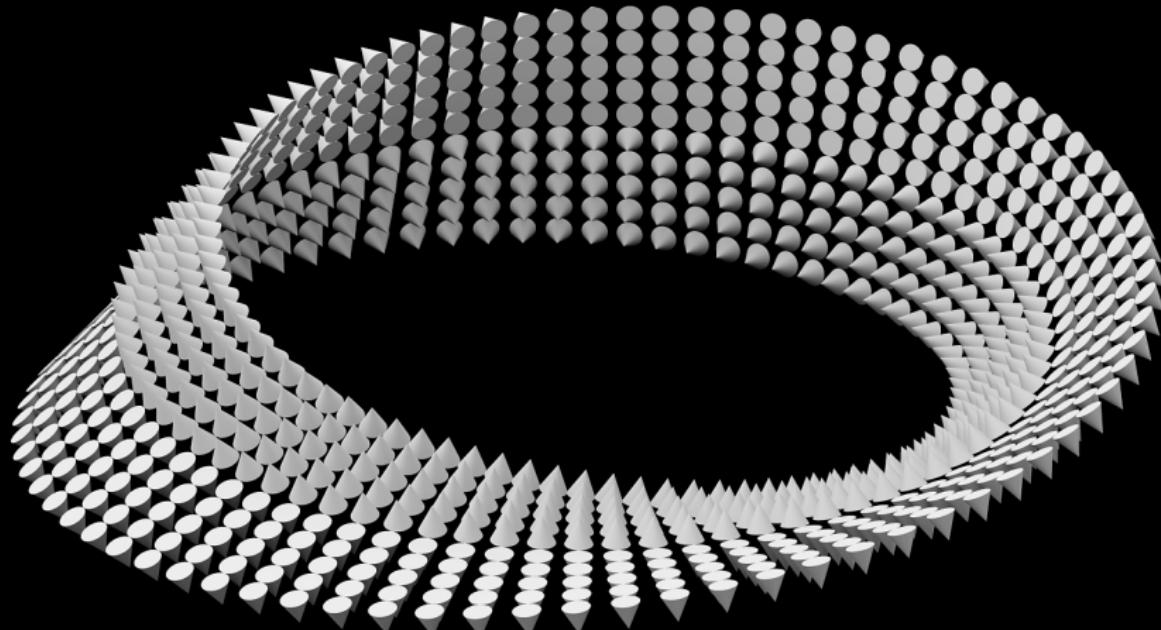
# Parametric Surfaces

## Tesselations



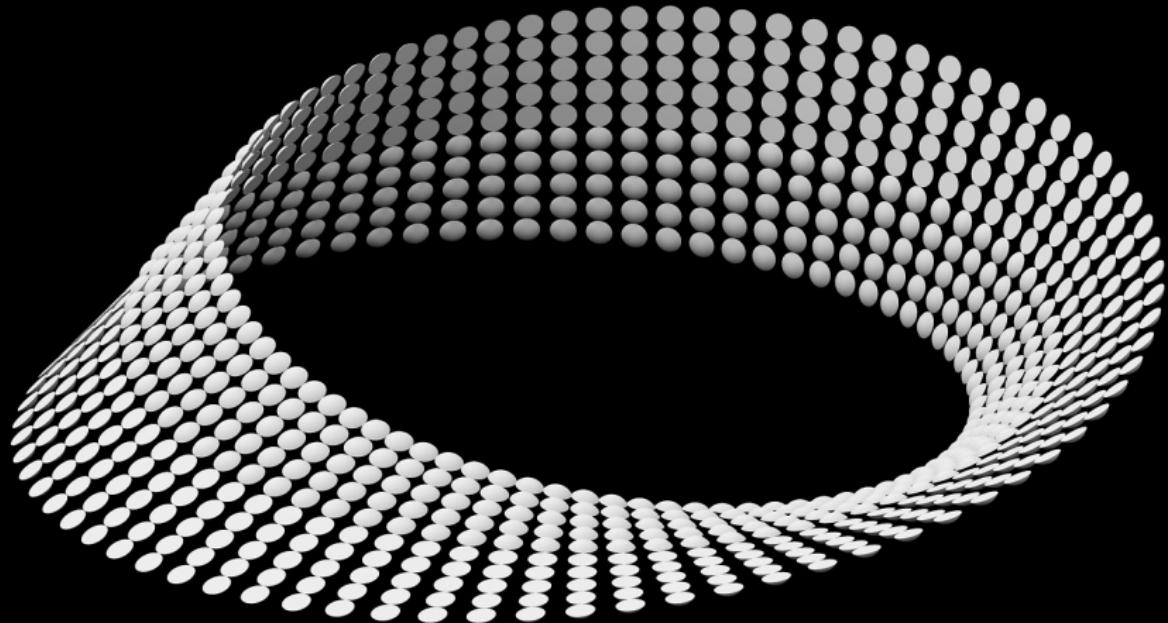
# Parametric Surfaces

## Tesselations



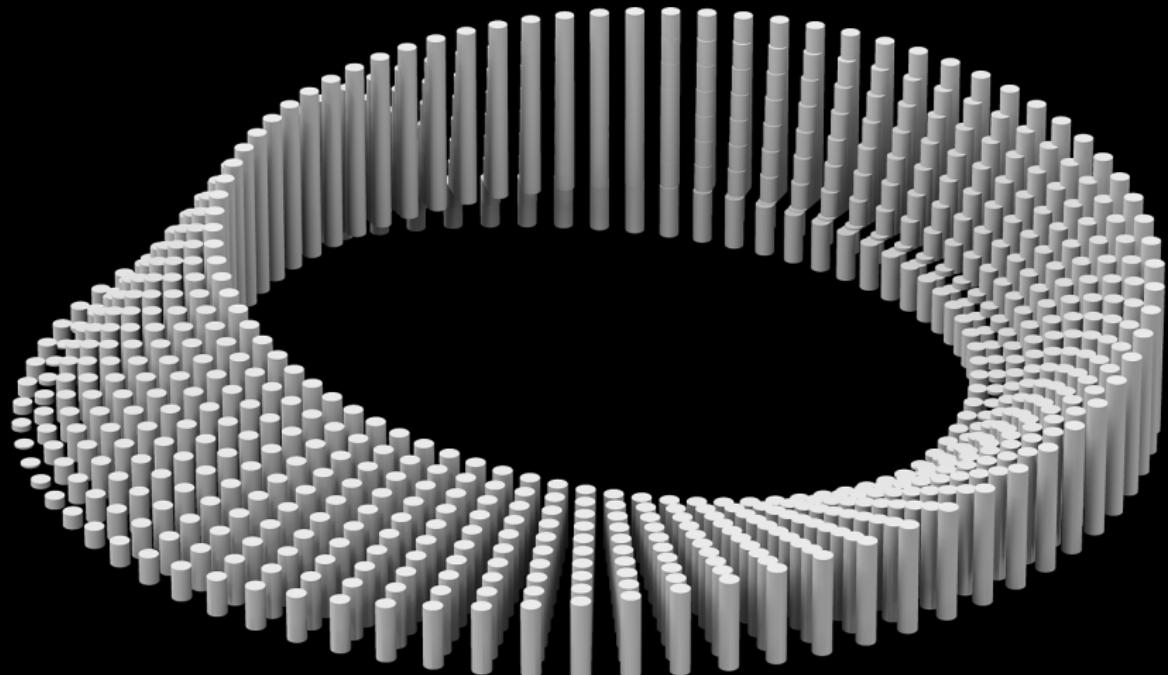
# Parametric Surfaces

## Tesselations



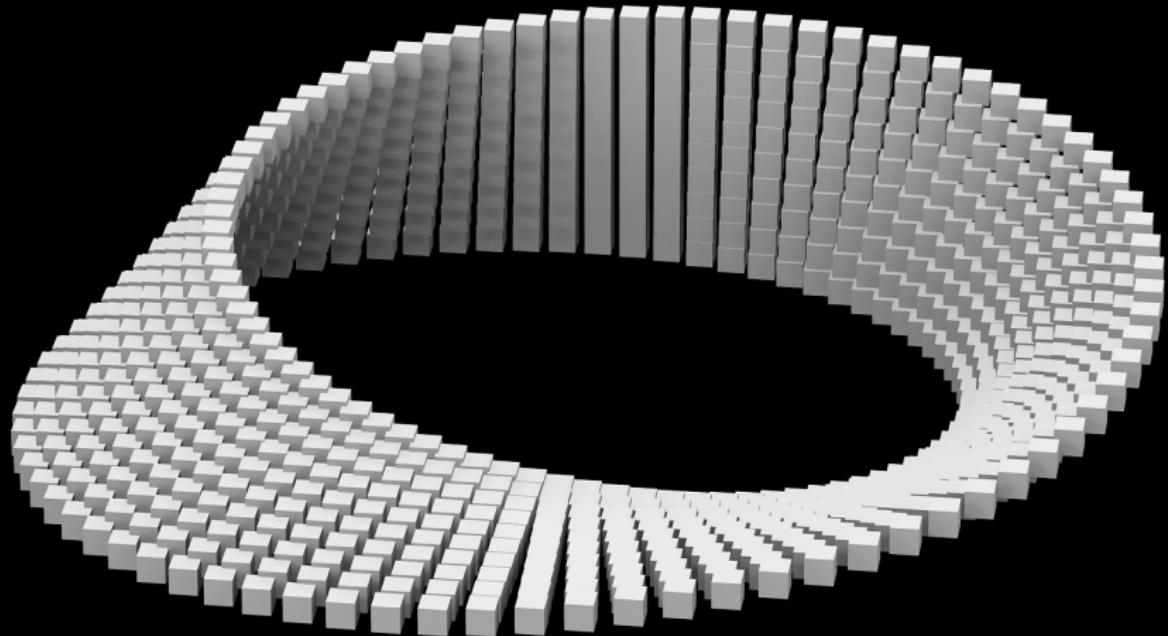
# Parametric Surfaces

## Tesselations



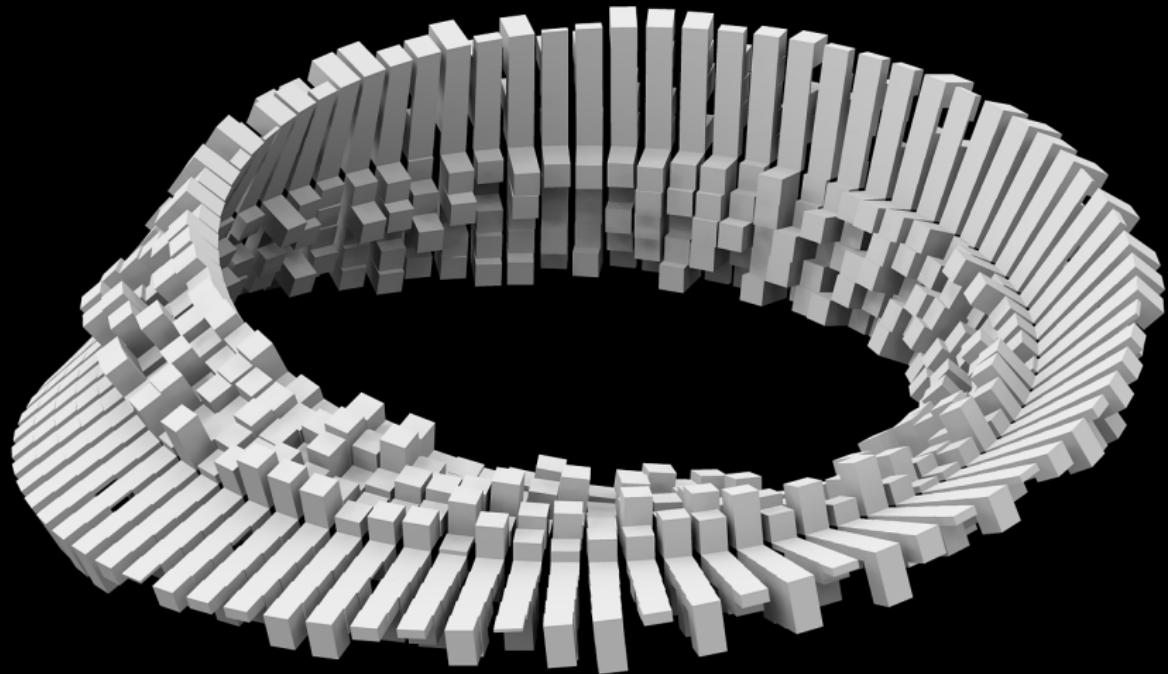
# Parametric Surfaces

## Tesselations



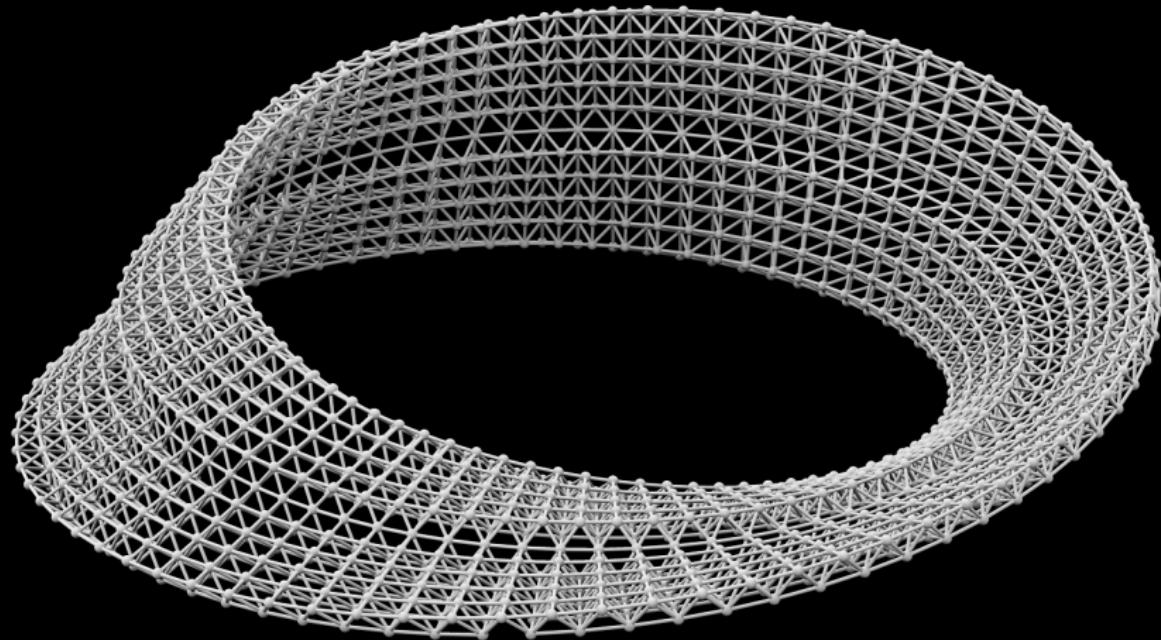
# Parametric Surfaces

## Tesselations



# Parametric Surfaces

## Tesselations



# Teaching

## Evaluation

- Project
- Exam
- Mini tests

# Teaching - Exam



INSTITUTO  
SUPERIOR  
TÉCNICO

Instituto Superior Técnico

Programação e Computação  
para Arquitectura – 2010/2011

Segundo Exame – 31/02/2011

Número: \_\_\_\_\_

Nome: \_\_\_\_\_

Responda a seu próprio nome em todas as folhas da prova. O tamanho das respostas deve ser limitado ao espaço fornecido para cada pergunta. Se tiver dúvidas de interpretação, faça explicações detalhadas e explique-as na sua resposta. Pode usar os versos das folhas para rascunhos. A prova tem 8 páginas e a duração é de 120 minutos. A cotação de cada questão encontra-se indicada entre parêntesis. Boa sorte.

Em todos os exercícios, pode usar todas as funções descritas na sobretira.

1. (2.0) Considere a seguinte definição AutoLisp para a função de Ackermann:

```
(defun ack (n m)
  (if (= n 0)
      m
      (+ (ack (- n 1) m)
          (ack (- n 1) (+ m 1)))))
```

(a) (0.5) A função apresentada é recursiva? Porque?

(b) (0.5) A função apresentada é de ordem superior? Porque?

(c) (0.5) Calcule o resultado de (ack 1 2).

(d) (0.5) Traduza a definição da função de Ackermann para a notação tradicional da matemática.

Número: \_\_\_\_\_

2

2. (2.0) Considere um modelo simplificado de um edifício baseado numa sequência de paralelepípedos empilhados, cuja dimensão é aleatória mas sempre sucessivamente mais pequena, tal como é ilustrado na seguinte imagem:



Considere que este modelo de edifício é parametrizado pelo número de paralelepípedos empilhados, pelas coordenadas do primeiro paralelepípedo e pelo comprimento, largura e altura do edifício. Os blocos de base tem exactamente o comprimento e largura especificados mas a sua altura deverá estar entre 20% e 80% da altura total do edifício. Os blocos seguintes estão centrados sobre os blocos imediatamente abaixo e possuem o comprimento e largura daquele sólido entre 70% e 80% das suas respectivas dimensões no bloco imediatamente abaixo. A altura destes blocos deverá ser entre 20% e 80% da altura restante do edifício. Dê-lhe uma função capaz de produzir este modelo de edifícios.

# Teaching - Exam

Número \_\_\_\_\_

3

3. (1.0) Defina a função `pontos_circunferencia` que, a partir das coordenadas de um ponto  $p$ , de um raio  $r$  e de um número de pontos  $n$ , devolve uma lista com as coordenadas de  $n$  pontos uniformemente distribuídos ao longo da circunferência de raio  $r$ , tal como se ilustra na imagem seguinte para  $n = 16$ .



4. (2.0) Defina a função `pontos_sinusoida_circular` de parâmetros  $p, r_i, r_o, r$  e  $n$  que computa uma lista com os pontos de uma curva fechada com a forma de uma sinusóide com  $n$  ciclos que se desenvolve num anel circular centrado no ponto  $p$  e delimitado pelos raios inferior  $r_i$  e exterior  $r_o$ , tal como se pode ver nos vários exemplos apresentados na seguinte figura onde, da esquerda para a direita, o número de ciclos é 12, 3 e 5.



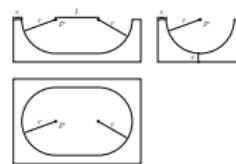
Número \_\_\_\_\_

4

5. (1.5) Pretende-se modelar uma banheira de pedra idêntica à que se apresenta na imagem seguinte:



Os parâmetros relevantes para a banheira encontram-se representados no alçado frontal, planta e alçado lateral apresentados na imagem seguinte:



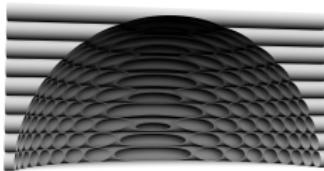
Defina uma função denominada `banheira` que constrói uma banheira idêntica à da figura anterior.

# Teaching - Exam

Número \_\_\_\_\_

5

6. (2,5) Considere a imagem seguinte:



A imagem representa um abrigo de forma paralelipípédica construído com tubos cilíndricos cortados de modo a que o espaço interior tenha a forma de um quarto de esfera. Note que os tubos mais grossos e mais extensos são definidos por duas sequências com o ponto cada vez mais próximo do centro da estrutura. Note ainda que o raio do quarto de esfera é igual à altura e largura do abrigo menos o raio de todos os cilindros.

Defina uma função que controla o abrigo a partir do centro da esfera, da altura do paralelipípedo e do número de tubos a colocar ao longo da altura.

Número \_\_\_\_\_

6

7. (2,0) Considere o seguinte esquema para os cabos verticais de uma ponte suspensa:



Os cabos verticais são modelados por cilindros de raio  $r$  (representados, no esquema, pelas linhas mais grossas e mais extensas) que são definidos por duas sequências com o ponto cada vez mais próximo da curva. A curva é concava para cima, ou seja, se  $x_0 < x_1 < x_2$ , a curva  $z = f(x)$  está ao longo da curva  $z = f(x)$ , a começar em  $(x_0, 0, f(x_0))$  e a acabar em  $(x_2, 0, f(x_2))$ .

Defina a função **cilindros-verticais** que recebe a função  $f$ , as abscissas  $x_0$  e  $x_1$ , o raio  $r$  e o número de cilindros  $n$  e constrói o modelo apresentado.

8. (2,0) Defina uma função denominada **estrela** que, a partir de um ponto  $p_0$  de um raio interno  $r_{in}$  de um raio exterior  $r_{ex}$  e de um número  $n$  de vértices no raio exterior, desenha estrelas como as apresentadas em seguida:



# Teaching - Exam

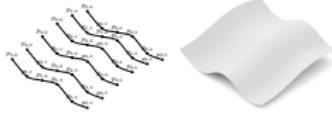
Número \_\_\_\_\_

7

9. (1.5) Considere uma superfície representada por um conjunto de coordenadas tri-dimensionais. Admita que essas coordenadas estão armazenadas numa lista de listas da forma:

```
( (p00, p01, ..., p0N),  
  (p10, p11, ..., p1N),  
  ...  
  (pM0, pM1, ..., pMN))
```

Defina a função **superficie-interpolação**, que recebe uma lista com a forma anterior, adiciona por cima uma lista de splines em que cada spline  $S_i$  passa pelos pontos  $p_{i,0}, p_{i,1}, \dots, p_{i,N}$  tal como se apresenta na imagem seguinte à esquerda e acaba por usar essa lista de splines  $S_0, S_1, \dots, S_M$  para fazer uma interpolação suave de secções, tal como se apresenta na imagem seguinte à direita:



10. (2.0) Considerem o seguinte elipsóide:



Número \_\_\_\_\_

8

- Um elipsóide é caracterizado pelas dimensões dos seus três raios ortogonais  $a, b$  e  $c$ . A sua equação paramétrica é:

$$\begin{aligned}x &= a \sin \vartheta \cos \phi \\y &= b \sin \vartheta \sin \phi \\z &= c \cos \vartheta\end{aligned}$$

$$-\frac{\pi}{2} \leq \vartheta \leq \frac{\pi}{2}, \quad -\pi \leq \phi \leq +\pi;$$

Defina a função **elipsóide** que produz o elipsóide a partir dos três raios  $a, b, c$  e ainda do número  $n$  de valores a usar ao longo de  $\vartheta$  e de  $\phi$ :

11. (1.5) Considere a seguinte superfície gerada por duas sinusóides que se desenvolvem em direções ortogonais:



Sabendo que a superfície oscila entre a altura máxima de 0.5 e a mínima de -0.5 e que  $x$  e  $y$  ambos variam no intervalo  $[0, 2\pi]$ , complete a seguinte expressão de modo a criar as coordenadas da superfície anterior:

```
(parametros = {a=0,b=0,  
lambda_a = 1,lambda_b = 1,x = V,  
y = W},  
u)
```

```
0.2*exp(2x)  
0.2*exp(2y))
```

# Teaching - Project

Projecto de  
Programação e Computação para Arquitectura  
Turning Torso

António Menezes Leitão  
14 de Fevereiro de 2008

## 1 Introdução

O Turning Torso é um edifício desenhado pelo arquitecto espanhol Santiago Calatrava. A Figura 1 mostra uma imagem deste edifício. Com uma altura de 100 metros divididos em 51 pisos, o Turning Torso é o mais alto edifício da Suécia.



Figura 1: Turning Torso. Fotografia de Jerzy Kocickiewicz.

O edifício é composto por nove "cubos" com cinco pisos cada. Cada cubo funciona como um edifício independente, sendo o espaço intermédio entre os cubos usado como espaço técnico para manutenção das fachadas.

Cada cubo sobre uma base à medida que garante altura sendo que o piso mais alto é sempre o mais pesado e mais grosso (no sentido dos pentes do relógio) relativamente ao piso mais baixo.

No exterior do edifício encontra-se um escorpião metálico, representado na Figura 2. Esta telesca é constituída por um tubo com braços horizontais e ilustrados anexados ao edifício. A telha está tocada para acompanhar a

# Teaching - Project

redução do edifício. Todo esse esqueleto faz o travamento horizontal do edifício e destina-se a aumentar a resistência do edifício aos ventos e a diminuir a sua vibração.



Figura 2: Turning Torso. Fotografia de Umberto Luparelli.

A estrutura de suporte do edifício é constituída por um núcleo cilíndrico de betão reforçado aço que abriga os ascensores e escadas de serviço bem como as estruturas hidráulicas. O núcleo tem 100 m de altura e 17 m de diâmetro, constante desde a base até ao topo, embora a sua espessura interior varie desde 2,5 m na base até 0,40 m no topo. Este núcleo é visível na Figura 3.

As estruturas de suporte são feitas de vigas de aço com seção elástica plana e 2250 juntas. Para acompanhar a torção, os painéis são inclinados para dentro no lado oeste do edifício e para fora no lado leste, variando a inclinação entre 0º e 7º. A inclinação lateral dos painéis é de 0º.

2



Figura 3: Turning Torso. Fotografia de Lars Tufveson.

Cada piso consiste de uma laje de secção rectangular arredondada acoplada a uma seção triangular. As lajes encostam e apóiam-se no núcleo central de suporte (veja Figura 4). O eixo da estrutura é o vértice das lajes, tal como se apresenta na Figura 4. Nas extremidades das lajes existem ancoragens de betão armado levadas de madeira a acompanhar a torção do edifício.

O edifício é de utilização mista, com os dois primeiros andares reservados para escritórios, os andares 3 a 10 para apartamentos e os andares 11 a 57 para apartamentos distribuídos por terraços e três diferentes tipologias, com áreas desde 45 a 190 metros quadrados.

Para mais informações sobre o edifício sugere-se a consulta dos seguintes sites:

- <http://www.vestasgroup.com/>
- <http://www.eurospine.com/projeto/turning-tower/>
- <http://www.acropole.com.br/turning-tower.html>
- <http://www.vestasgroup.com/projeto/turning-tower/111.aspx?cspid=451-1.aspx>
- <http://www.acropole.com.br/turning-tower/index.html>
- <http://www.vestasgroup.com/lospages/realizar-a-pedra-pivoteante.aspx?l1id=1>

3

# Teaching - Project



Figura 4: Turning Torso. Planta de um piso.

## 2. Trabalho a Desenvolver

O projeto é para ser realizado em grupos de dois alunos ou, excepcionalmente, de um só aluno.

O projeto deve ser feito na escrita de um programa Auto-Lisp capaz de gerar o Turning Torso. O seu programa deverá ser suficientemente parametrizado para que se possam explorar variações em torno das ideias fundamentais da obra do Calatrava. Em particular, deverá ser possível experimentar variações nas seguintes parâmetros:

- número de pisos por bloco;
- número de blocos no edifício;
- ângulo de rotação de cada piso;
- altura de cada piso;
- dimensões do núcleo;
- dimensões das paredes exteriores;
- curvatura das paredes exteriores;
- dimensões das barras da trépida

Naturalmente, importa que todos os parâmetros definidores da geometria do edifício estejam bem identificados para que seja possível alterá-los facilmente.

Para permitir uma avaliação intercalar do projeto, este será decomposto em duas fases com as seguintes datas de entrega:

4



Figura 5: Turning Torso. Fotografia de Wanshan.

1. Primeira fase: a entregar até às 24:00 do dia 9 de Dezembro de 2007.

2. Segunda fase: a entregar até às 24:00 do dia 21 de Dezembro de 2007.

O trabalho a entregar em cada fase é detalhado em seguida.

### 2.1 Primeira Fase

Nesta fase pretende-se que implemente um programa Auto-Lisp capaz de gerar os pisos do edifício, sem necessidade de incluir quaisquer elementos internos excepto o núcleo cilíndrico de suporte. Este núcleo deverá obedecer à geometria definida na planta. Os pisos devem ser definidos de forma parametrizada para permitir experimentar variações na sua forma. Não é necessário mas é recomendável que a geração dos pisos possa afectar-lhe um determinado ângulo de rotação.

Até ao final da prazo de entrega desta fase deverá andar a página da cadeira no Fórum para submeter o seu trabalho por via eletrónica. O trabalho a submeter deverá ser composto por um arquivo **.3DP** ou **.TAS-.Q3** contendo:

5

# Teaching - Project



Figura 6: Turning Torso. Fotografia de Lisa Liu.

- um ou mais ficheiros Auto Lisp com o código desenvolvido para o projeto.
- uma ou mais imagens (JPG, EPS, PDF, PNL, DWG, etc.) com vistas do edifício geradas.

O código desenvolvido deverá estar escrito num estilo que facilite a leitura e que dispense, na medida do possível, excessivos comentários. Tudo deverá ser incluído, não para dizerem o que o código já diz claramente, mas para documentar o que é feito, as funcções principais e, eventualmente, algumas partes menos claras dos programas.

Esta entrega será avaliada mas não será atribuída qualquer nota, servindo apenas para informar os alunos do andamento dos seus trabalhos e para discutir quaisquer dúvida que tenham quando ao sucesso ou insucesso das ações que foram executadas.

6



Figura 7: Turning Torso. Fotografia de John Liu.

## 2.2 Segunda Fase

Na segunda fase devem-se ir tão longe quanto possível na modelação malha do Turning Torso. Em particular deverá ser possível modular as paredes do edifício com os painéis de alumínio e suas juntas e deverá incluir-se a trilha de insolação.

Ah! ao final do prazo de entrega desta fase deverá andar à página da cadeia no Fórum para submeter o seu trabalho por via electrónica. O trabalho a submeter deverá ser composto por um arquivo ZIP ou TAR.GZ contendo:

- um ou mais ficheiros Auto Lisp com o código desenvolvido para o projeto.
- um documento PDF com um relatório do projeto.

## 2.3 Código

O código deve ser escrito de forma a facilitar a leitura e dispensando excessivos comentários. É sempre preferível ter código mais claro com poucos comentários do que ter código obscuro com muitos comentários.

O código deve ser dividido em módulos, dividido em funções com responsabilidades específicas e reduzidas. Cada módulo deverá ter um curto comentário a descrever o seu objectivo.

O código só é avaliado pelo corpo docente polo que devem incluir informações sobre qual a função principal e qual o significado dos seus parâmetros.

7

# Teaching - Project



Figura 8: Turning Torso. Fotografia de Kenttlin Andmussen.

## 2.2.2 Relatório

O relatório do projeto consiste de um documento PDF com uma descrição dos trabalhos desenvolvidos, explicando as decisões de programação tomadas e ilustrado com imagens e/ou animações que demonstram o resultado. As imagens devem demonstrar a versatilidade dos meios para a geração de geometrias alternativas para o edifício.

## 3 Avaliação

Os critérios de avaliação incluem:

- A qualidade das soluções desenvolvidas.
- A clareza dos programas desenvolvidos.
- A capacidade de geração de geometrias alternativas.

- A qualidade do relatório.

Em caso de dúvida, o corpo docente poderá exigir explicações sobre o funcionamento do projecto desenvolvido, incluindo eventuais demonstrações.

## 4 Plágio

Considera-se plágio o uso de quaisquer fragmentos de programas que não tenham sido fornecidos pelos docentes da disciplina. Não se considera plágio o uso de bibliografia.

Esta disciplina segue regras muito rígidas relativamente ao plágio. Quaisquer projetos que sejam considerados plagiados serão anulados, independentemente de quem plagiou e de quem tiver sido plagiado, independentemente de se o plágio é intencional ou não. O uso de código de terceiros em qualquer projeto não deverá ser impedimento para a troca salutar de ideias e para a normal camaradagem e ensinada que deve existir entre colegas. Contudo, aqueles que copiam códigos fragmentados de programas sob pena de quem os recebe não os entender e se limitar a plagiar-los com maior ou menor esforço de "camuflagem."

## 5 Notas Finais

Não se esqueça da Lei de Murphy.

# Results

# Results

# Results

# Results

# Results

# Results

# Practice

## Different Goals

- Automation
- Generative Design
- Research

## Common Needs

- CAD Application
- Programming Language

# Practice

## Alternatives

- ArchiCad
  - GDL
- AutoCAD
  - Visual Basic
  - C++
  - AutoLisp
- Rhino
  - RhinoScript
  - Grasshopper
  - Python
- Microstation
  - VBA
  - C++
  - Java

# Practice

## Alternatives

- ArchiCad
  - GDL
- AutoCAD
  - Visual Basic
  - C++
  - AutoLisp
- Rhino
  - RhinoScript
  - Grasshopper
  - Python
- Microstation
  - VBA
  - C++
  - Java

Incompatible

# Rosetta

The screenshot shows the DrRacket IDE interface with a file named "p.rkt". The code defines functions to generate shapes and a shape cloud.

```
#lang racket

(require rosetta)

(backend "rhino")

(define (randomize-sshape d1 d2 sr shape-fn)
  (let* ((r (random-interval d1 d2))
         (th (random-interval 0 (* pi 2)))
         (fi (random-interval 0 pi))
         (p (sph r th fi)))
    (move p (shape-fn (- sr r) p)))

(define (shape-cloud d1 d2 r n shape-fn)
  (for/list ((i (range n)))
    (randomize-sshape d1 d2 r shape-fn)))

(shape-cloud
  4
  5
  5
  600
  (λ (r n) (box-normal r r r n)))

#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
>
```

Determine language from source ▾ 604:2

# Rosetta

The screenshot shows the DrRacket IDE interface with a window titled "p.rkt - DrRacket". The menu bar includes File, Edit, View, Language, Racket, Insert, Tabs, Help, and several tool buttons: Check Syntax, Debug, Macro Stepper, Run, and Stop. The code editor contains the following Racket code:

```
#lang racket

(require rosetta)

(backend "rhino")

(define (randomize-sshape d1 d2 sr shape-fn)
  (let* ((r (random-interval d1 d2))
         (th (random-interval 0 (* pi 2)))
         (fi (random-interval 0 pi))
         (p (sph r th fi)))
    (move p (shape-fn (- sr r) p)))

(define (shape-cloud d1 d2 r n shape-fn)
  (for/list ((i (range n)))
    (randomize-sshape d1 d2 r shape-fn)))

(shape-cloud
  4
  5
  5
  600
  (λ (r n) (box-normal r r r n)))

#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
>
```

The status bar at the bottom indicates "Determine language from source" and a character count of "604:2".

## Programming Environment

- Editor
- Debugger
- Listener

# Rosetta - Multiple Back-Ends

The image displays two software windows side-by-side, illustrating the Rosetta project's ability to handle multiple back-ends.

**Left Window (DrRacket):**

- Title:** p.rkt - DrRacket
- Menu Bar:** File, Edit, View, Language, Racket, Insert, Tabs, Help
- Status Bar:** p.rkt ▾ (define ...) ▾ Check Syntax, Debug, Macro Stepper, Run, Stop
- Code Area:**

```
#lang racket

(require rosetta)

(backend "rhino")

(define (randomize-sphere d1 d2 sr shape-fn)
  (let* ((r (random-interval d1 d2))
         (th (random-interval 0 (* pi 2)))
         (fi (random-interval 0 pi))
         (p (sph r th fi)))
    (move p (shape-fn (- sr r) p)))

(define (shape-cloud d1 d2 r n shape-fn)
  (for/list ((i (range n)))
    (randomize-sphere d1 d2 r shape-fn)))

(shape-cloud
  4
  5
  5
  600
  (λ (r n) (box-normal r r r n)))

#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>

>
```
- Status Bar:** Determine language from source ▾ 604.2

**Right Window (Rhinoceros):**

- Title:** Untitled - Rhinoceros (Evaluation) - [Top]
- Menu Bar:** File, Edit, View, Curve, Surface, Solid, Mesh, Dimension, Transform, Tools, Analyze, Render, Help
- Status Bar:** Creating meshes... Press Esc to cancel, Command
- View:** A 3D view showing a sphere composed of numerous small, randomly oriented cubes, representing the generated mesh.
- Toolbars and Panels:** Various toolbars and panels for 3D modeling are visible along the top and sides of the Rhinoceros interface.

# Rosetta - Multiple Back-Ends

p.rkt - DrRacket

File Edit View Language Racket Insert Tabs Help

p.rkt ▾ (define ...) ▾ Check Syntax Debug Macro Stepper # Run Stop

```
#lang racket

(require rosetta)

(backend "autocad")

(define (randomize-sshape d1 d2 sr shape-fn)
  (let* ((r (random-interval d1 d2))
         (th (random-interval 0 (* pi 2)))
         (fi (random-interval 0 pi))
         (p (sph r th fi)))
    (move p (shape-fn (- sr r) p)))

(define (shape-cloud d1 d2 r n shape-fn)
  (for/list ((i (range n)))
    (randomize-sshape d1 d2 r shape-fn)))

(shape-cloud
  4
  5
  5
  600
  (λ (r n) (box-normal r r r n)))

#<autocad-shape>
#<autocad-shape>
#<autocad-shape>
#<autocad-shape>
#<autocad-shape>
```

Determine language from source ▾ 603:2

Autodesk AutoCAD

Drawing1.dwg Type a keyword or phrase

Home Help

Steering Wheels Views World Opacity Viewports Windows

N E W S TOP WCS

REGEN Command: MODEL

1.100 Medium Detail Cut Plane: 1400.00

# Rosetta - Multiple Front-Ends

The image shows two software windows side-by-side, illustrating the Rosetta project's ability to interface with multiple front-end environments.

**Left Window (DrRacket):** A code editor window titled "crater-tessellation.js - DrRacket". The code is written in Racket, defining functions for creating rectangles, craters, and wavy skins, and applying tessellation to crater features.

```
require("rosetta");
require("racket");
backend("autocad");

function rectangleLine(p1, p2, p3, p4) {
    return join(spline(list(p1, p2)),
                spline(list(p2, p3)),
                spline(list(p3, p4)),
                spline(list(p4, p1)));
}

function crater(c, n, p1, p2, p3, p4, radius) {
    return loft(
        list(rectangleLine(p1, p2, p3, p4),
            move(pPlusp(c, (pStar(n, 0.1 * radius))),
                 circleN(radius, n)),
            move(pPlusp(c, (pStar(n, 0.8 * radius))),
                 circleN(0.7 * radius, n)),
            move(pPlusp(c, (pStar(n, -0.1 * radius))),
                 circleN(0.4 * radius, n))));
}

function wavySkin() {
    return makeSimpleSurface(
        function(u, v) {
            return xyz(u,
                       v,
                       0.4*sin(u+v)+0.1*abs(u-1)*sin(v-1));
        },
        makeUvDomain('closed', 0, 'closed', 3,
                     'closed', 0, 'closed', 6));
}

function craterTessellation(skin, nU, nV) {
    return map(function(patch) {
        var n1 = patchObject(patch, nUMin/1, nVMin/1);
        ...
    });
}
```

**Right Window (AutoCAD Architecture 2011):** A 3D modeling interface showing a complex surface model with numerous circular features. The interface includes toolbars for Steering, Views, World, Visu..., Viewports, and Windows, and a status bar at the bottom.

# Rosetta - Multiple Front-Ends

The image displays two software windows side-by-side, illustrating the Rosetta project's ability to interface with multiple front-end environments.

**Left Window (DrRacket):** This window shows an Racket script named "atomium.rsp". The code defines a function "atomium" that generates a molecular model. It uses the "rosetta" and "racket" libraries, and the "rhino" backend. The function creates a series of points (p0 through p8) and spheres (pts0 through pts1), then uses mapcar to apply a lambda function to each pair of points. This lambda function moves one point to another, creates cylinders between them, and rotates the second point. The final step is to flatten the resulting list of atoms.

```
#lang autolisp

(require "rosetta")
(require "racket")
(backend "rhino")

(define atomium (re 1 xc / p0 p1 p2 p3 p4 p5 p6 p7 p8 pts0 pts1)
  (setq p0 (xyz 0 0 0)
    p1 (xyz (- 1) (- 1) (- 1))
    p2 (xyz (+ 1) (- 1) (- 1))
    p3 (xyz (+ 1) (+ 1) (- 1))
    p4 (xyz (- 1) (+ 1) (- 1))
    p5 (xyz (- 1) (- 1) (+ 1))
    p6 (xyz (+ 1) (- 1) (+ 1))
    p7 (xyz (+ 1) (+ 1) (+ 1))
    p8 (xyz (- 1) (+ 1) (+ 1))
    pts0 (list p1 p2 p3 p4)
    pts1 (list p5 p6 p7 p8))
  (list (mapcar '(lambda (p) (move p (sphere xc)))
    (cons p0 (append pts0 pts1)))
    (mapcar '(lambda (pa pb) (cylinder-pp rc pa pb))
      pts0
      (cdr (append pts0 pts0)))
    (mapcar '(lambda (pa pb) (cylinder-pp rc pa pb))
      pts1
      (cdr (append pts1 pts1)))
    (mapcar '(lambda (pa pb) (cylinder-pp rc pa pb))
      pts0
      pts1)
    (mapcar '(lambda (pa pb)
      (list (cylinder-pp rc pa p0)
        (cylinder-pp rc p0 pb)))
      pts0
      (cddr (append pts1 pts1)))))

  (map (λ rotate-pp ux one) (flatten (atomium 9 32 1.5))))
```

**Right Window (Rhinoceros):** This window shows a 3D perspective view of the generated molecular model. The model consists of eight spheres connected by a network of lines, forming a complex polyhedron-like structure. The Rhinoceros interface includes a toolbar, a command palette, and a coordinate system with axes (X, Y, Z).

# Rosetta - Portability

The image shows two side-by-side DrRacket windows. The left window, titled "arabic.js - DrRacket", contains Racket code for an L-system interpreter. The right window, titled "lsystem.rkt - DrRacket", contains Racket code for a system that provides rules for L-systems.

**arabic.js - DrRacket**

```
require('rosetta');
require("lsystem.rkt");
backend('autocad');

var myRules = rules(rule("F", "F+FFF+F"));

function interpretLSystem(str, p, rho, phi, dphi, r) {
  for (var i = 0; i < str.length(); i++) {
    switch (str.charAt(i)) {
      case '+':
        phi+=dphi;
        break;
      case '-':
        phi-=dphi;
        break;
      case 'F':
        var p0 = p,
            p1 = Pluspol(p, rho, phi);
        p=p1;
        evaluate(move(p, sphere(r)));
        evaluate(cylinderPp(r, p0, p1));
        break;
    }
  }
}

interpretLSystem(new String(
  repeatApplyRules(myRules, "F", 4),
  zero, 1, 0, Math.PI/3, 0.2);
```

**lsystem.rkt - DrRacket**

```
#lang racket

(provide rules rule repeat-apply-rules)

(define rules list)
(define rule cons)

(define (apply-rules rules str)
  (string-append*
   (map (lambda (c)
          (cdr (or (assoc (string c) rules)
                   (cons c (string c))))))
   (string->list str)))

(define (repeat-apply-rules rules str n)
  (if (= n 0)
      str
      (repeat-apply-rules
       rules
       (apply-rules rules str) (- n 1))))
```

# Rosetta - Portability

arabic.js - DrRacket

File Edit View Language JavaScript Insert Tabs Help

arabic.js ▾ Debug Check Syntax Run Stop

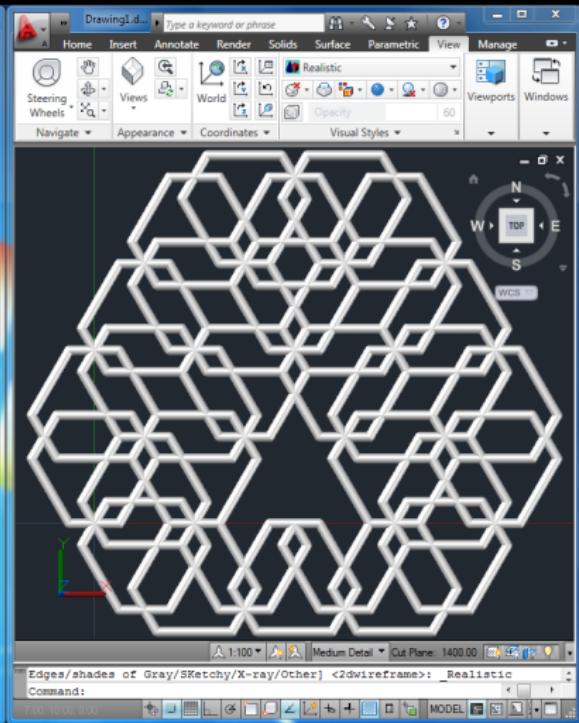
```
require('rosetta');
require("lSystem.rkt");
backend('autocad');

var myRules = rules(rule("F", "F+FFF+F"));

function interpretLSystem(str, p, rho, phi, dphi, r) {
  for (var i = 0; i < str.length(); i++) {
    switch (str.charAt(i)) {
      case '+':
        phi+=dphi;
        break;
      case '-':
        phi-=dphi;
        break;
      case 'F':
        var p0 = p,
            p1 = Pluspol(p, rho, phi);
        p=p1;
        evaluate(move(p, sphere(r)));
        evaluate(cylinderPp(r, p0, p1));
        break;
    }
  }
}

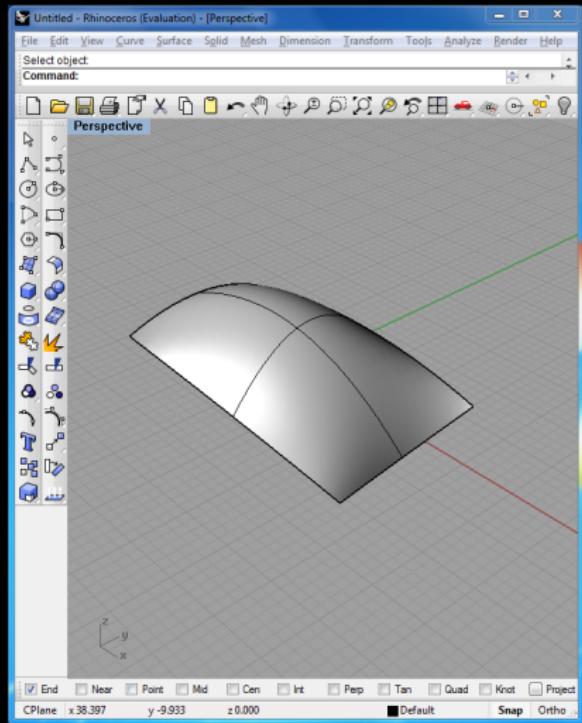
interpretLSystem(new String(
  repeatApplyRules(myRules, "F", 4),
  zero, 1, 0, Math.PI/3, 0.2);
```

JavaScript ▾

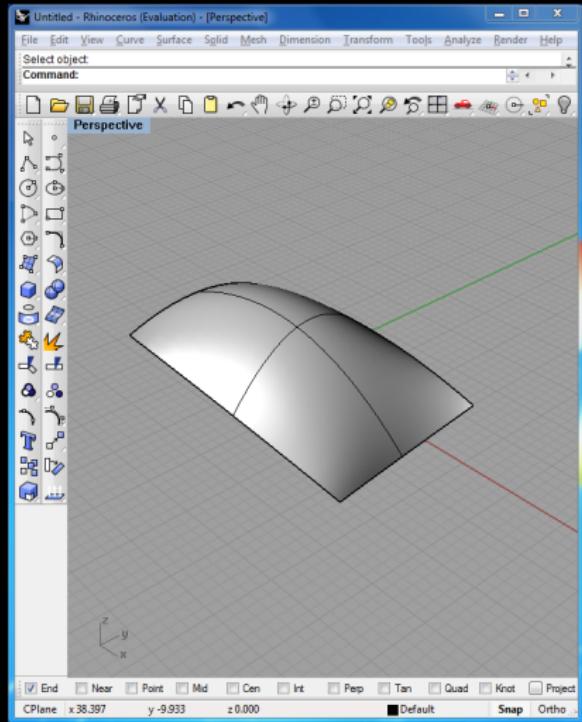


# Rosetta - Interactivity

# Rosetta - Cross-CAD Applications

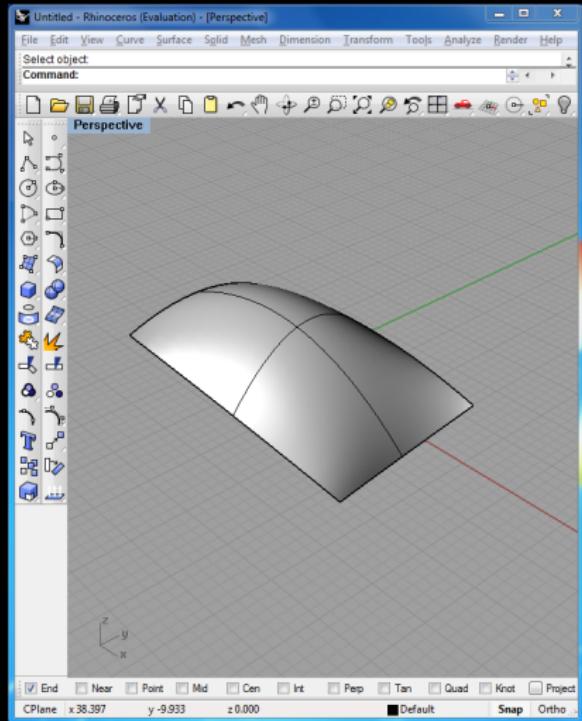


# Rosetta - Cross-CAD Applications



(backend "rhino")

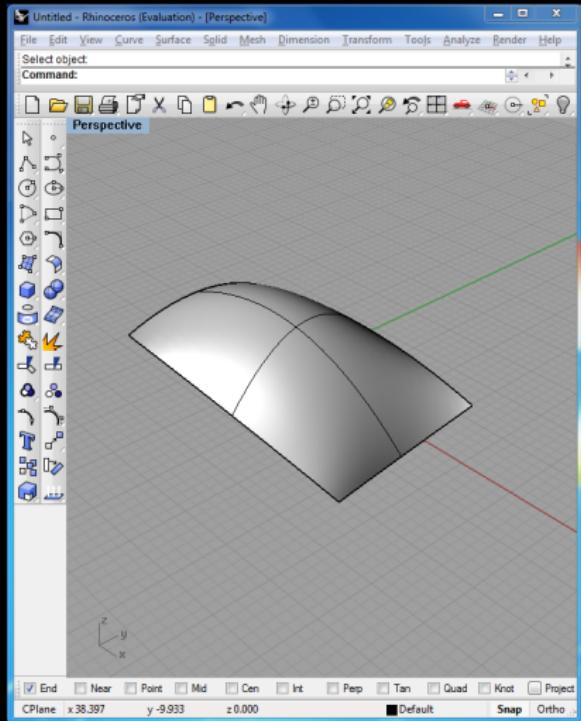
# Rosetta - Cross-CAD Applications



(backend "rhino")

```
(define surf  
  (get-object "Select surface"))
```

# Rosetta - Cross-CAD Applications

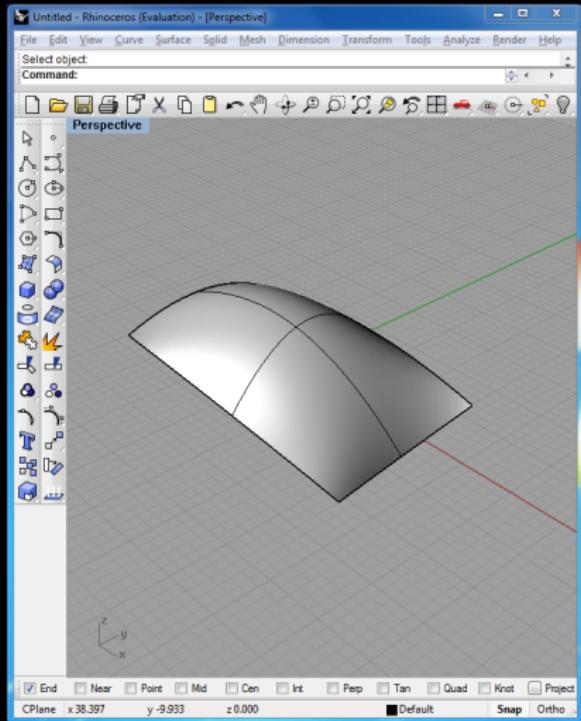


(backend "rhino")

```
(define surf  
  (get-object "Select surface"))
```

(backend "autocad")

# Rosetta - Cross-CAD Applications



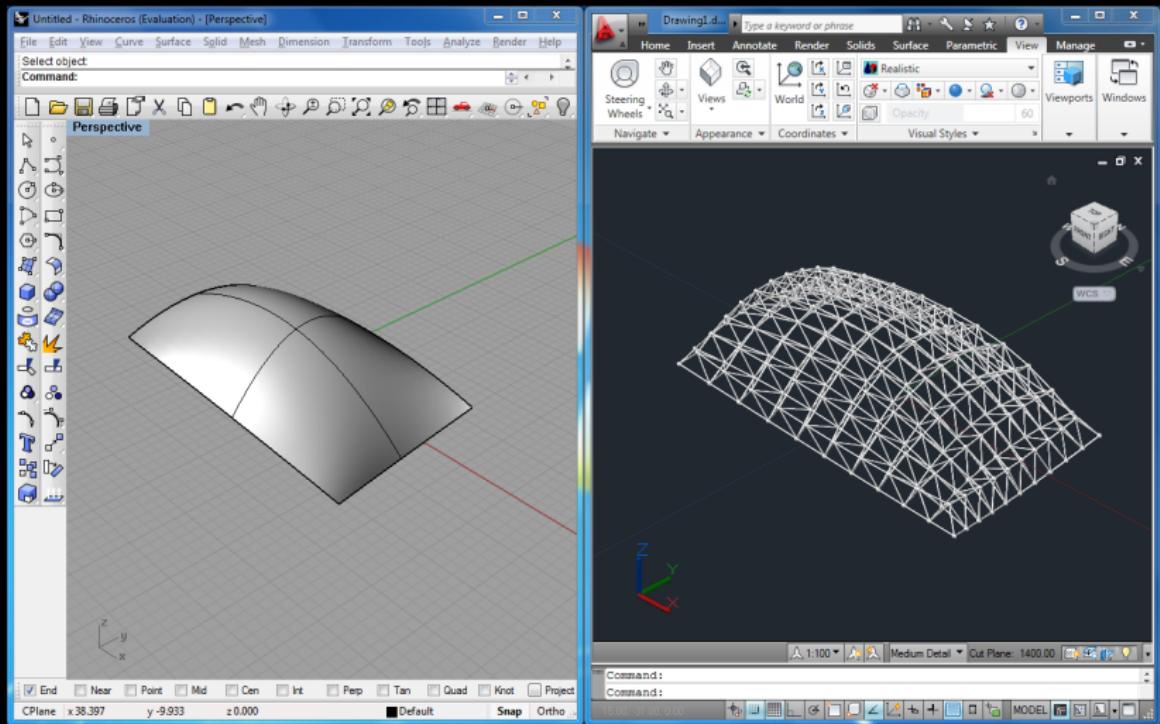
(backend "rhino")

```
(define surf  
  (get-object "Select surface"))
```

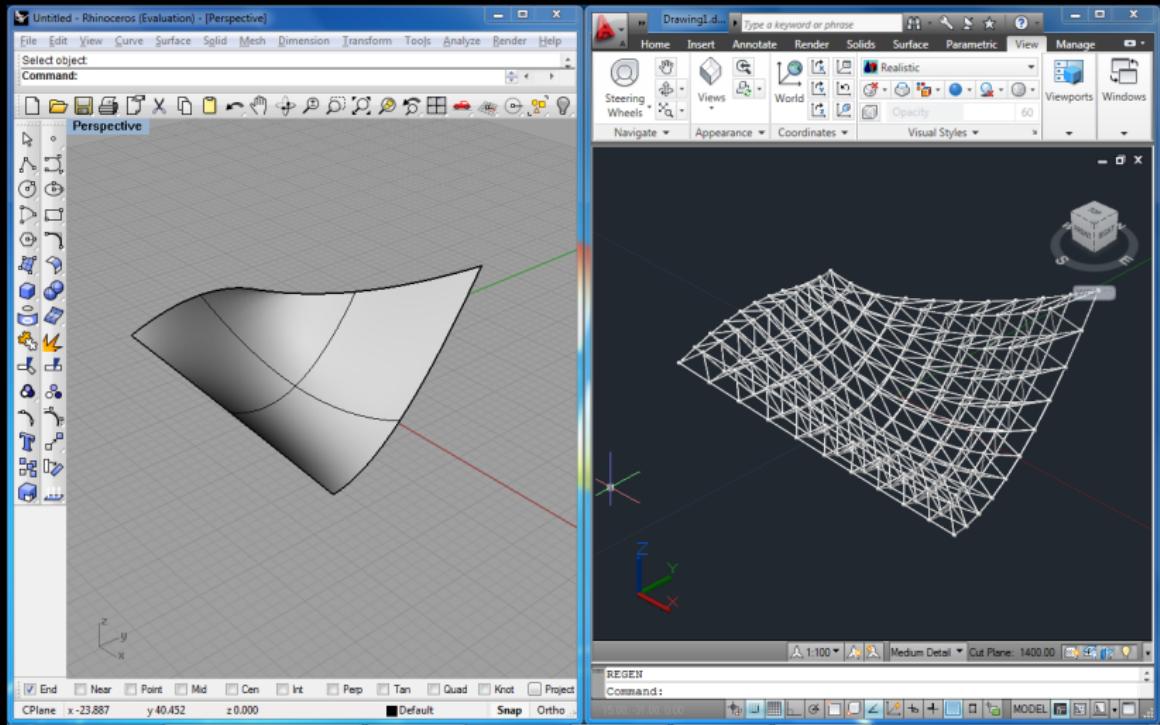
(backend "autocad")

(truss-from-surface surf)

# Rosetta - Cross-CAD Applications



# Rosetta - Cross-CAD Applications



# Conclusion

*The architect must be a prophet... a prophet in the true sense of the term... if he can't see at least ten years ahead don't call him an architect*

*Frank Lloyd Wright*

# Conclusion

*The architect must be a prophet... a prophet in the true sense of the term... if he can't see at least ten years ahead don't call him an architect*

*Frank Lloyd Wright*

Programming will be a fundamental tool for Architecture

# Conclusion

*The architect must be a prophet... a prophet in the true sense of the term... if he can't see at least ten years ahead don't call him an architect*

*Frank Lloyd Wright*

Programming will be a fundamental tool for Architecture

Thank You

# Conclusion

*The architect must be a prophet... a prophet in the true sense of the term... if he can't see at least ten years ahead don't call him an architect*

*Frank Lloyd Wright*

Programming will be a fundamental tool for Architecture

Thank You

Questions?