

# Computer Animation

## Natural Phenomena SS 19



Prof. Dr. Charles A. Wüthrich,  
Fakultät Medien, Medieninformatik  
Bauhaus-Universität Weimar  
caw AT medien.uni-weimar.de

# Groups of objects

- Multiple objects can form groups
- Among them, we will introduce
  - Particles
  - Flocks
  - Autonomous agents

# Controlling groups of objects

- A *particle system* is a large collection of individual elements which taken together represent a conglomerate object
- The „global“ behaviour of the particles is called *emergent behaviour*
- This can be used both for particle systems (which usually have more individuals) and for *flocking*
- Flock members have a more sophisticated behaviour than a simple element of particle system
- While particle systems behave according to physics, flocking particles add some intelligence to the behaviour of the individuals
- The more intelligence is added, the more the element moves in a more interesting way, and the more it shows *autonomous behaviour*

# Particle systems

- In a particle system, due to the no of its elements, simplified assumptions are made
- Typical assumptions are
  - Particles do not collide among themselves
  - Particles do not cast indiv. shadows, but the aggregate may do
  - Particles only cast shadows on the rest of the environment, not among themselves
  - Particles do not reflect light, each is modeled as a point light source
- Often particles are modeled as having a finite life span
- To avoid dull behaviour, often randomness is added
- When a particle system is computed, the following steps are taken:
  - Generate new particles born this frame
  - Initialize attributes of new particle
  - Remove dying particles
  - Animate active particles
  - Render them

# Particle generation

- Particles are usually generated according to a stochastic process
  - At each frame, a random number  $r_p$  of particles is generated
  - Generation has a user specified distribution centered at the desired number of particles per frame
  - $r_p = \text{ave} + \text{Rand}(\text{seed}) \cdot \text{range}$  where ave is the desired average and range is the desired variation range
- Sometimes it may be convenient to have this random function as a function of time, i.e. to make the number of desired particles increase in time
- If the particles are used to model a fuzzy object, then the area of the screen covered by the object  $A_s$  is used to control the number of particles

$$r_p = \text{ave} + \text{Rand}(\text{seed}) \cdot \text{range} \cdot A_s$$

# Particle attributes

- Attributes of the particles are typically
  - Position
  - Velocity
  - Shape parameters
  - Color
  - Transparency
  - Lifetime
- At each frame, the lifetime of each particle is decremented by one until it reaches zero
- During lifetime, particles are animated (position, velocity, shape, color, transparency)
- At each frame, forces on the particles are computed
- These result in an acceleration, which determines a velocity
- Also other attributes may be a function of time
- Rendering is often done modeling them as a point light source adding color to the pixel
- This to avoid particles to contribute to lighting computations

# Flocks

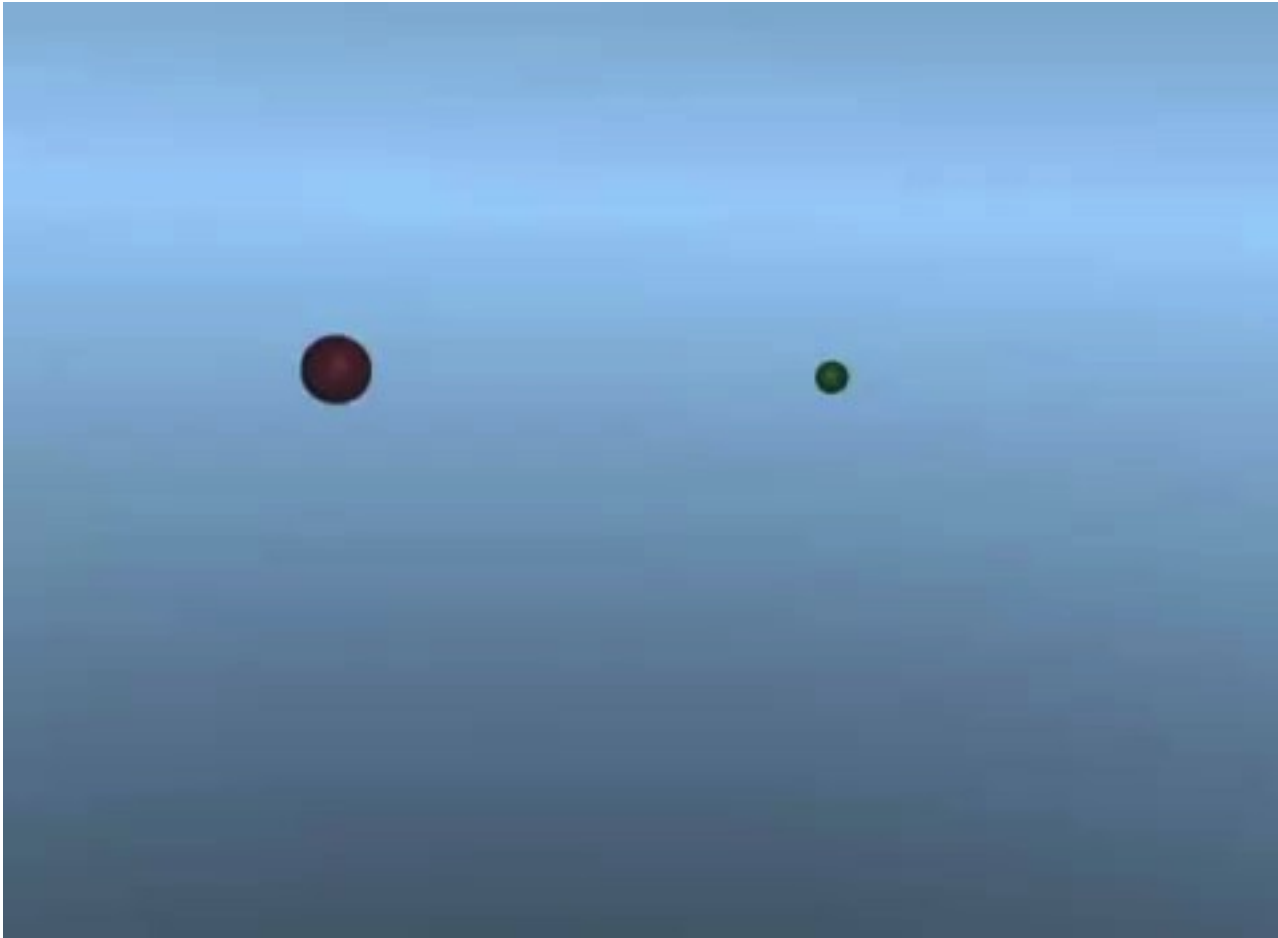
- Here the number of members is smaller
- But each member has some intelligence and simple physics (avoid collision, gravity, drag)
- Aggregate behavior emerges from the members (emergent behavior)
- Each member is called a boid
- Two forces govern flock behavior:
  - collision avoidance: both with other boids and with obstacles
  - Motion has some random parameter to keep it from looking regular
  - flock centering: the boid tries to be a flock member
  - Flock centering keeps together the flock but does not have to be absolute, otherwise flocks cannot split around objects

# Flocks: local behavior

- Controlling locally the behavior is what one aims at
- Three processes may be modeled:
  - Physics: similar to particle with gravity, collision detection and response
  - Perception of the environment: each boid views its direct neighbors and obstacles directly in front
  - Reasoning and reaction to determine the behavior
  - Additionally velocity matching is added (each boid tries to match the speed of its neighbours)
- Global control is either applied to all boids or to a group leader
  - In this case the boids follow the leader
- The leader role can be rotated among boids in time
- Usually all this is implemented as three controllers which are prioritized in the following order: collision avoidance, velocity matching and flock centering



# Flocks



# Flock complexity

- The major problem with flocks is the fact that processing complexity is  $N^2$ .
- Even if interactions are allowed only with  $k$  nearest neighbors, those have to be found
- One way to find efficiently is to perform a 3d bucket sort and then check adjacent buckets for neighbors
- Of course, efficiency depends on the bucket size:
  - The more buckets, the less boids per bucket
- Another way of doing it is through message passing, where each boid informs the flock of its whereabouts

# Collision avoidance

- There are several ways to avoid collisions
  - The simplest way is adding a repelling force around an object
  - However, this looks weird as the boid keeps attempting to aim at the repelling surface and constantly gets blown away
  - Another method computes if the boid trajectory hits the surface and starts a steering behavior
  - Quite complicated is the simulation of a splitting flock around an obstacle, since a balance has to be found between collision avoidance and flock cohesion

# Autonomous behaviour

- Recently, authors concentrated in learning and simulating how “intelligent” behaviour can be implemented.
- Needs-goals prioritization
- Some concentrated on “learning simulations” and life games



# Autonomous behaviour

- Modeling intelligent behaviour is a complex task
- Autonomous behaviour models an object knowing about its environment
- This can become as complicated as one wants
- Usually applied to animals, but also to people, cars on a road, planes, or soldiers in a battle
- Knowledge of the environment is provided by providing access to the environment geometry
- Subjective vision can be achieved by rendering the environment from the point of view of the object
- Internal state is modeled by **intentions** = the urge to satisfy a need
- High level goals can be decomposed in single low level tasks (levels of behaviour)
- Internal state and knowledge of the environment are input to the reasoning unit, which produces a strategy (=what needs to be done)
- Such strategy is turned into a sequence of actions by the planner, and actions are turned into movement
- If intentions are competing, they must be prioritized
- Look at this link:  
<http://www.youtube.com/watch?v=pqBSNAOsMDc&feature=related>

# Natural Phenomena

- One of the most challenging parts of animation systems is trying to model nature
- Many techniques and special mathematics is needed to do so
- Since nature is complex, it is often very time consuming to simulate nature
- Typical simulations include plants, water, clouds

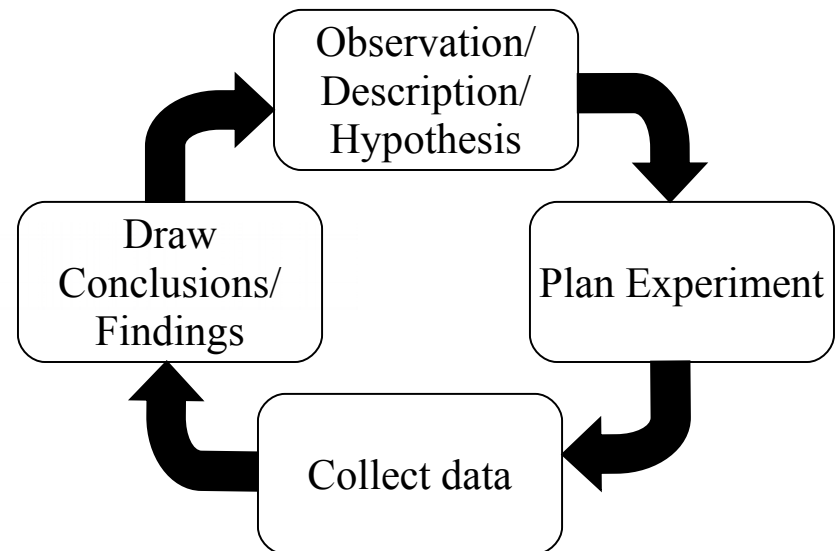


# Plants

- Plants possess an extraordinary complexity
- Lots of work was done on modeling the static representation of plants (Prusinkiewicz & Lindenmayer)
- Their observation was that plants develop according to a recursive branching structure
- If one understands how recursive branching works, one can model its growing process
- We are not going to go into detail on botanical terminology: I believe we can assume it as known

# The scientific method

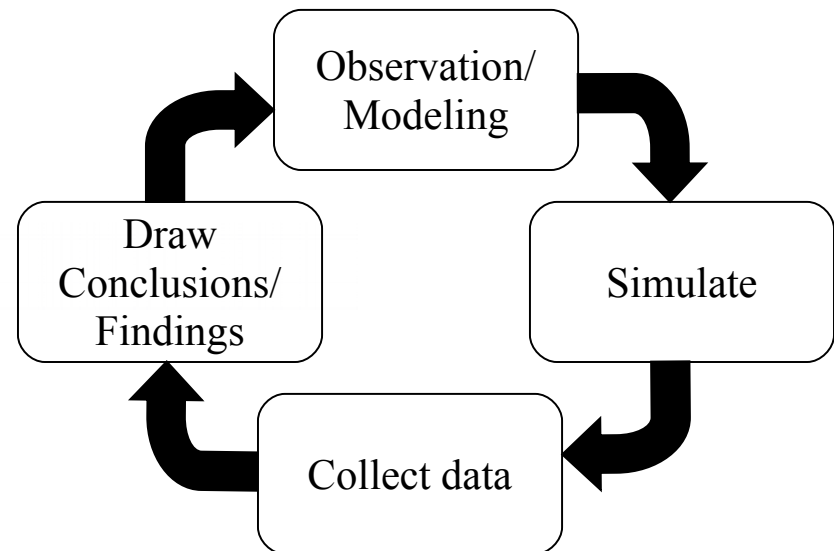
- But first we want to introduce something called the scientific method
- In the course of centuries the scientific community has perfected a method for making scientific progress
- Method based on a cycle
  - Observe, make hypothesis
  - Prepare experiment
  - Execute experiment and collect data
  - If data confirms hypothesis, publish
  - Otherwise, make new hypothesis and restart



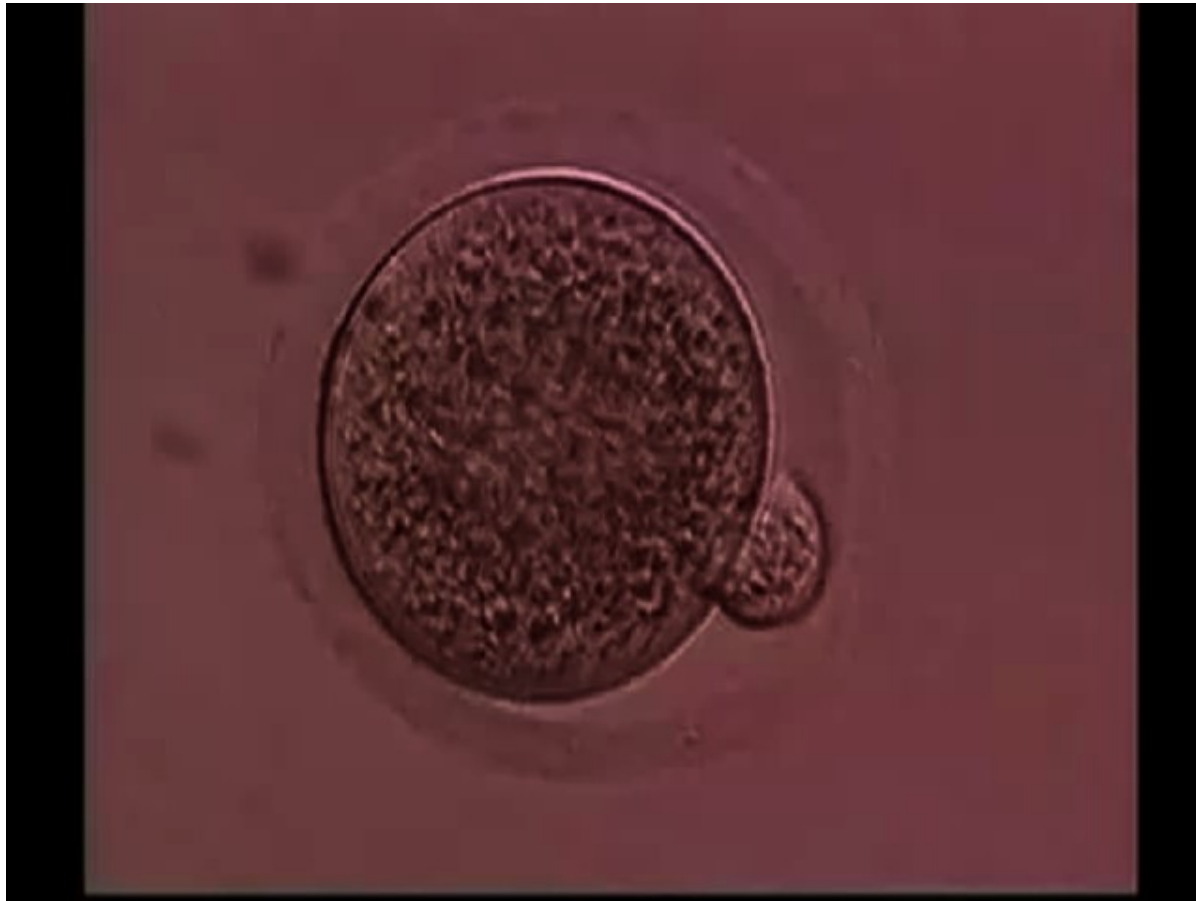


# What about modeling?

- Models are abstract representations of processes, entities or phenomena.
- Models allow simulations,
- These can be tested against what nature does to verify such models.
- Confirm that the model and laws applied in the simulation work properly
- Like scientific method, only that here the model is tested to see if it is a good representation



# An example

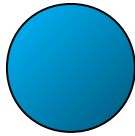


# A second example

- Who can describe what happened?

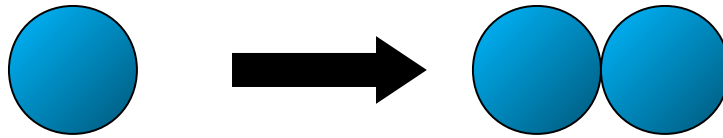


# A second example



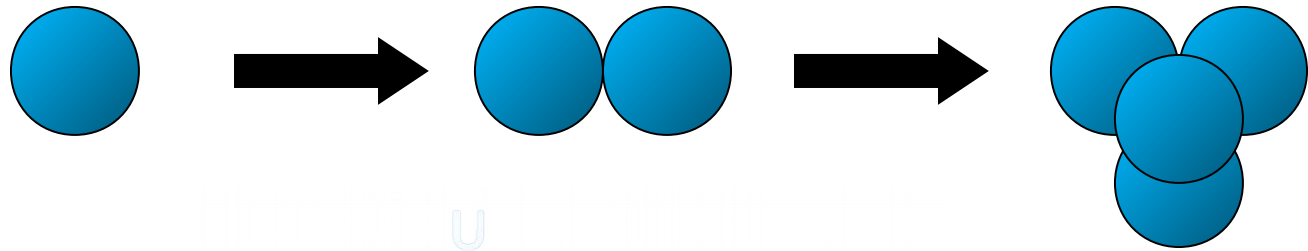
- Let us start from a single cell:
  - Anyone know how it evolves?

# A second example



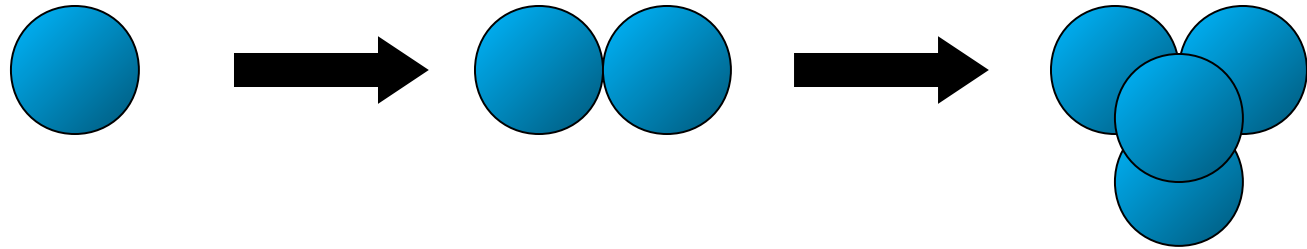
- Correct! Cells subdivide, first in two...

# A second example



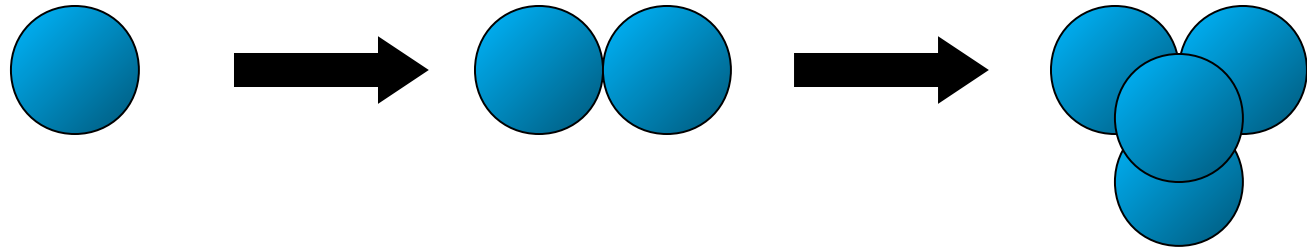
- Correct! Cells subdivide, then in four...

# A second example



- Let us now try to find some mathematical way of simulating this...

# A second example



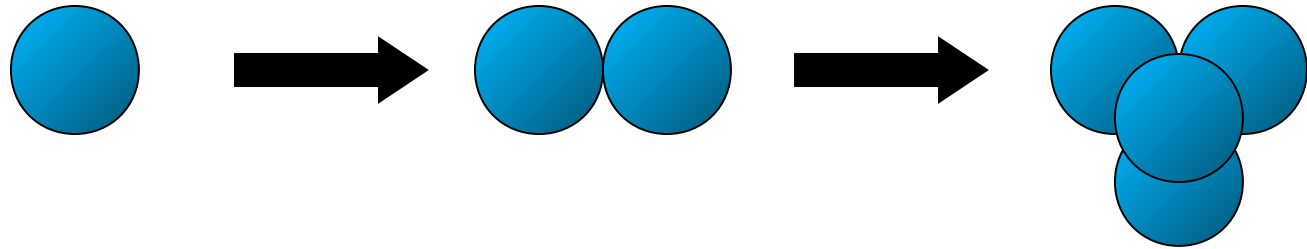
- Let us now try to find some mathematical way of simulating this...

U

- Anyone help me?

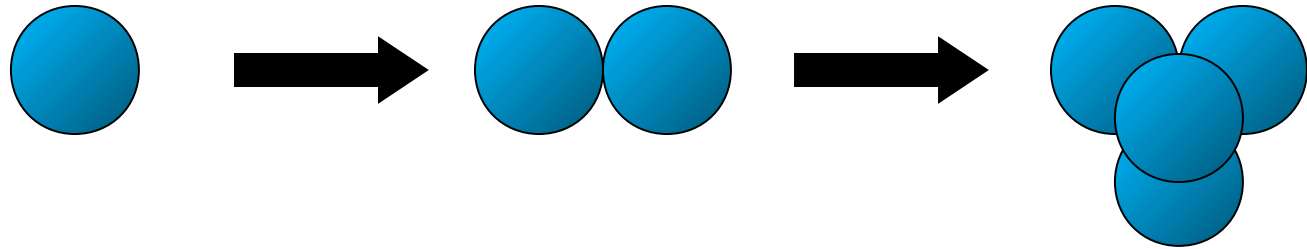


# A second example



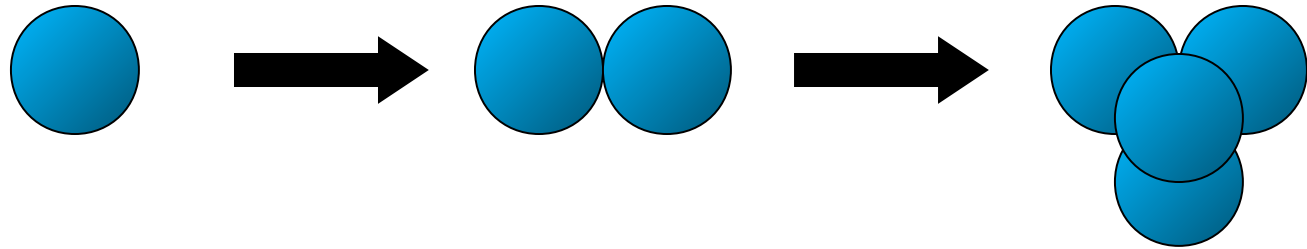
- Let us decide first which characteristics are unimportant for cell development:
  - Is spatial position important?

# A second example



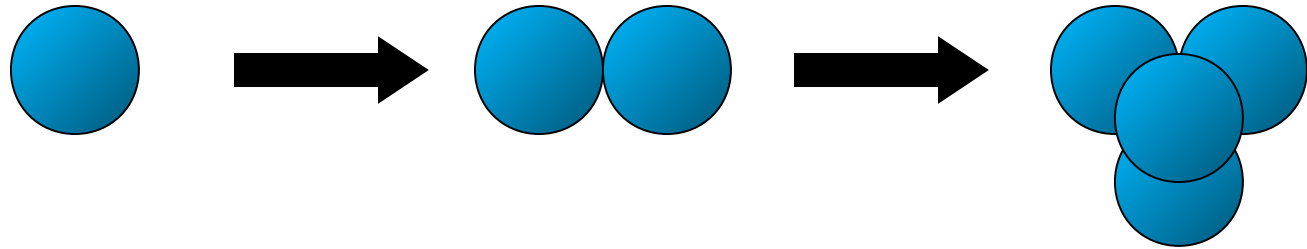
- Let us decide first which characteristics are unimportant for cell development:
  - Is spatial position important?  
In first approximation, NO!
  - From the point of view of cell growth, is the time axis important?

# A second example



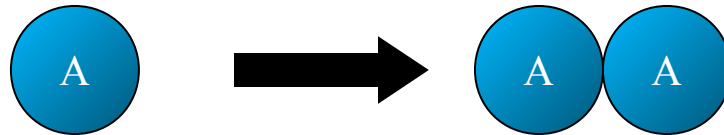
- Let us decide first which characteristics are unimportant for cell development:
  - Is spatial position important?  
In first approximation, NO!
  - From the point of view of cell growth, is the time axis important?  
Only at certain key events!

# A second example



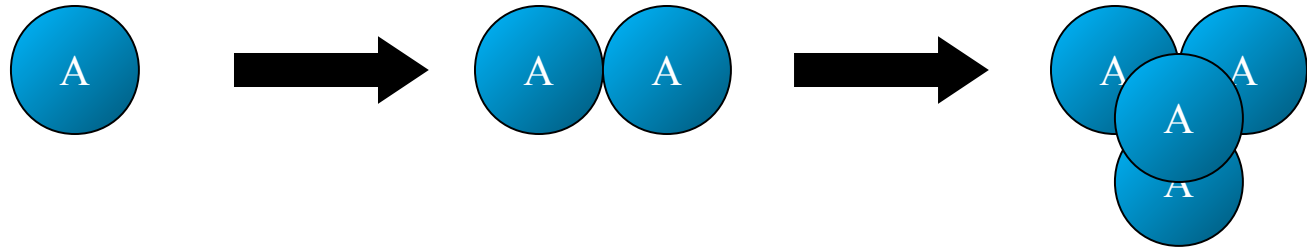
- Let us decide first which characteristics are unimportant for cell development:
  - Is spatial position important?  
In first approximation, NO!
  - From the point of view of cell growth, is the time axis important?  
Only at certain key events!
- This means that time can run at key events, i.e. in a non-continuous way

# Abstraction



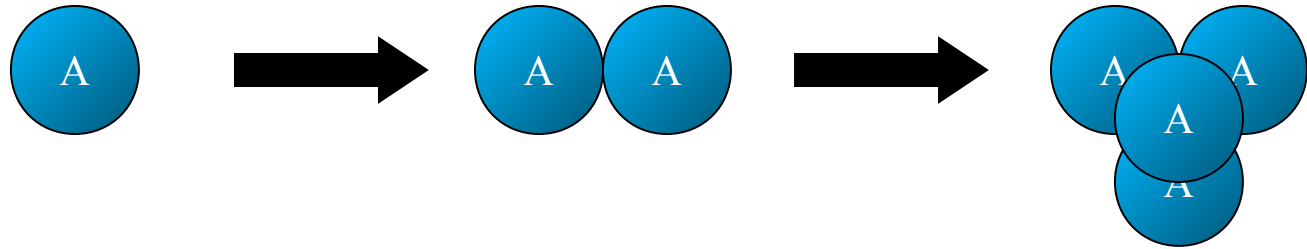
- So let us reduce this to the essential:
  - At a certain instant, a cell splits in two
- Let us call the cell A for the purpose of abstraction:
  - Cell A duplicates, so after A comes two times A, i.e. AA  
In symbols,  $A \rightarrow AA$

# Abstraction



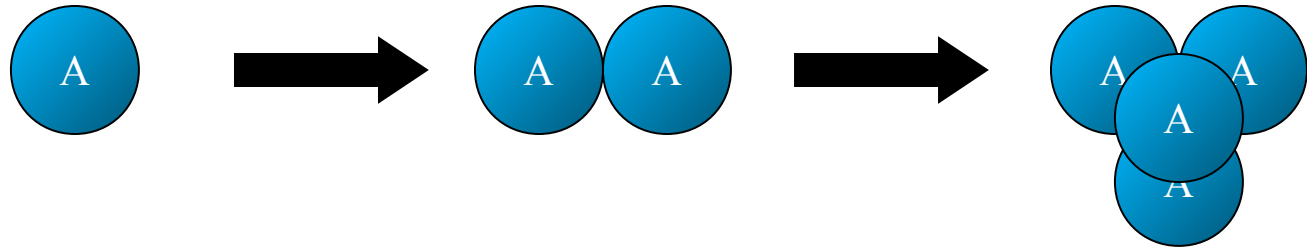
- So let us reduce this to the essential:
  - At a certain instant, a cell splits in two
- Let us call the cell A for the purpose of abstraction:
  - Cell A duplicates, so after A comes two times A, i.e. AA  
In symbols,  $A \rightarrow AA$
  - Then the two new A cells duplicate again, so AA becomes AAAA  
In symbols,  $AA \rightarrow AAAA$

# Abstraction



- So let us reduce this to the essential:
  - At a certain instant, a cell splits in two
- Let us call the cell A for the purpose of abstraction:
  - Cell A duplicates, so after A comes two times A, i.e. AA  
In symbols,  $A \rightarrow AA$
  - Then the two new A cells duplicate again, so AA becomes AAAA  
In symbols,  $AA \rightarrow AAAA$
  - And so on, and on, and on.....

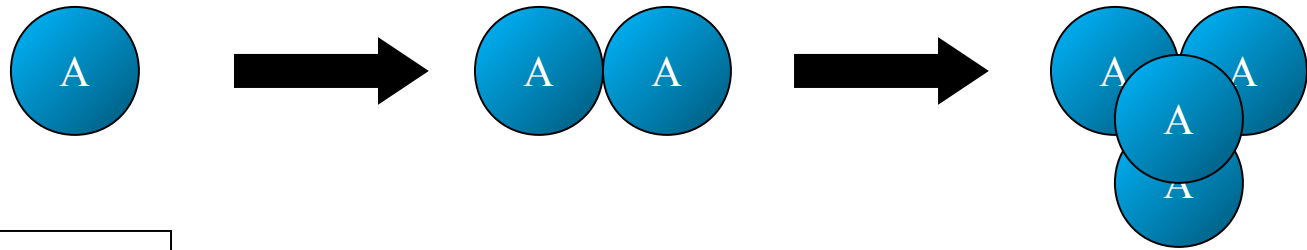
# Modeling the math



- What do we have here?
  - Cells, actually of one type
  - Does it have to be only one type?
  - Who says so?
- In fact, if we want to abstract a bit, we have to consider more than one type of cells....
- Thus, an element of a finite set  $\Sigma = \{A_1, A_2, \dots, A_n\}$



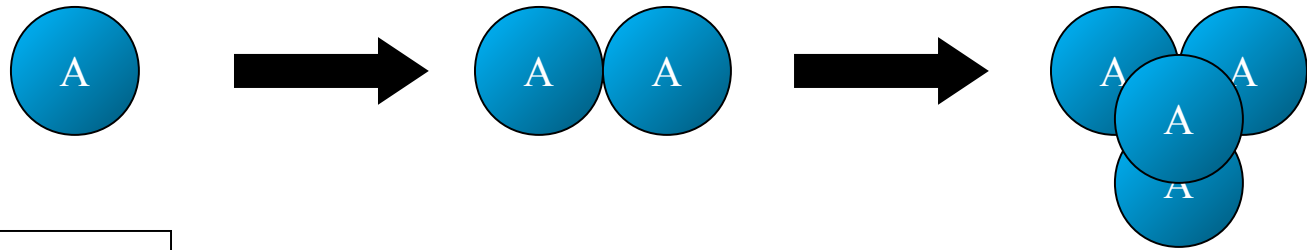
# Modeling the math



$$\Sigma = \{A_1, A_2, \dots, A_n\}$$

- So, we have a finite set of symbols  $\Sigma = \{A_1, A_2, \dots, A_n\}$   
Then what?
- Since we have generated juxtapositions of letters, the sequences of letters we obtain are nothing else than *juxtapositions* of letters of the alphabet  $\Sigma$ .
- Mathematically, this is called the set  $\Sigma^*$ .

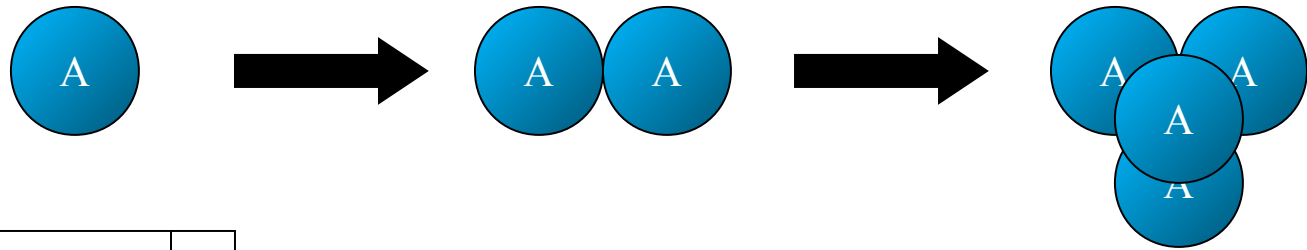
# Modeling the math



$$\Sigma = \{A_1, A_2, \dots, A_n\}$$

- So, we have a finite set of symbols  $\Sigma = \{A_1, A_2, \dots, A_n\}$ , and the juxtapositions of its letters  $\Sigma^*$   
Then what?
- How did the video start?

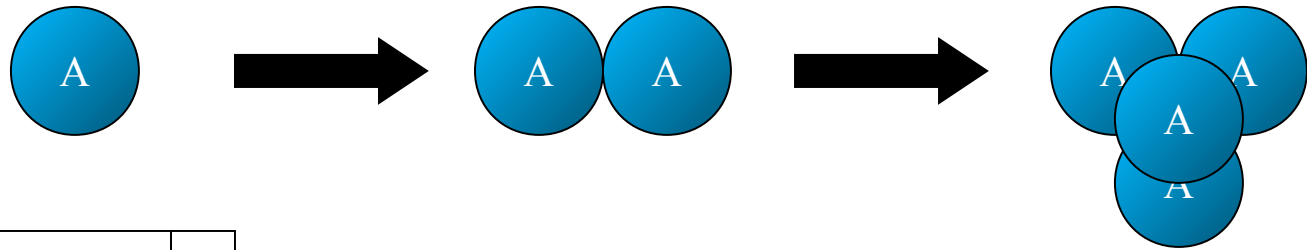
# Modeling the math



$\Sigma = \{A_1, A_2, \dots, A_n\}$	$\alpha$
-------------------------------------	----------

- So, we have a finite set of symbols  $\Sigma = \{A_1, A_2, \dots, A_n\}$  and  $\Sigma^*$ .  
Then what?
- How did the video start?
  - With ONE cell.
  - And who is this ONE cell? Nothing else than an element of  $\Sigma$
  - It does not have to be one, one could start from a string
  - This initial string is called the *axiom*
  - And obviously,  $\alpha \in \Sigma^*$

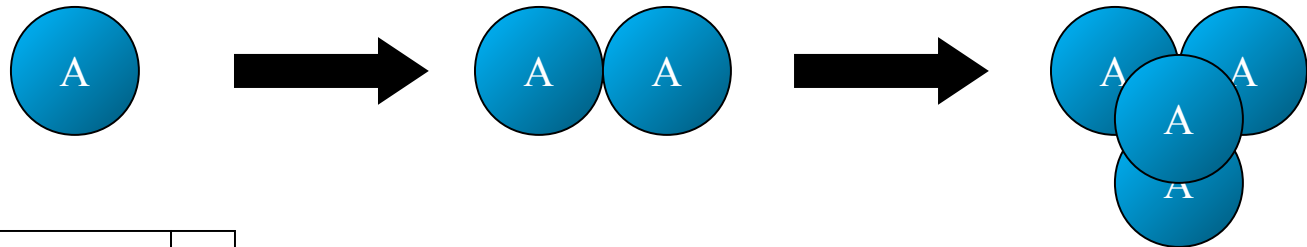
# Modeling the math



$\Sigma = \{A_1, A_2, \dots, A_n\}$	$\alpha$
-------------------------------------	----------

- So, we have a finite set of symbols  $\Sigma = \{A_1, A_2, \dots, A_n\}$ , and an axiom  
Then what?
- What was the cell split thing again?

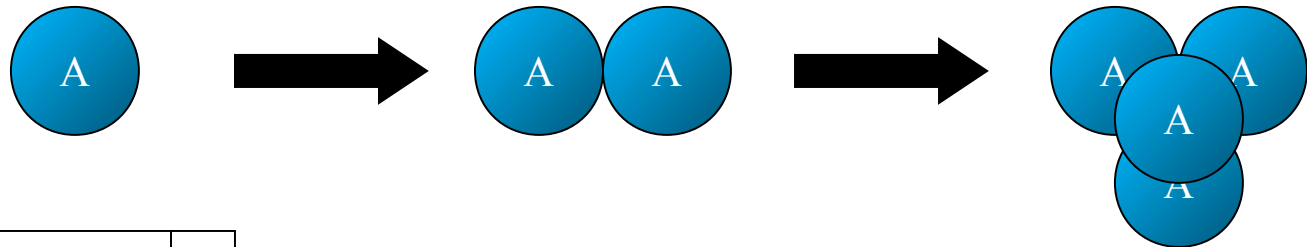
# Modeling the math



$\Sigma = \{A_1, A_2, \dots, A_n\}$	$\alpha$
-------------------------------------	----------

- So, we have a finite set of symbols  $\Sigma = \{A_1, A_2, \dots, A_n\}$ , and an axiom  
Then what?
- What was the cell split thing again?
  - In this case, it substitutes one symbol with itself two times.
  - Basically it substitutes to one symbol another one, or two of them, or three..

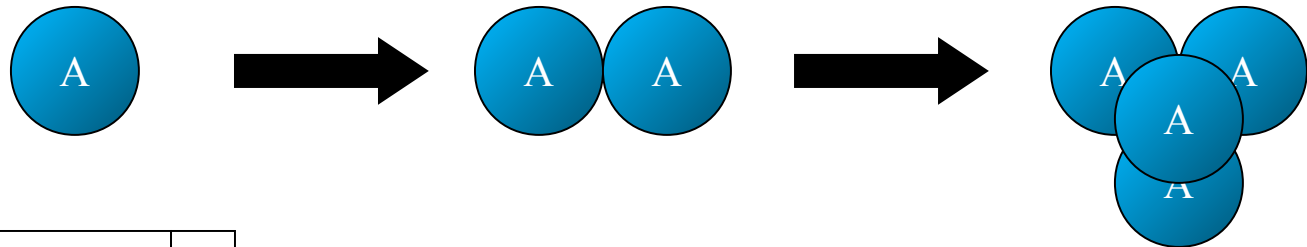
# Modeling the math



$\Sigma = \{A_1, A_2, \dots, A_n\}$	$\alpha$
-------------------------------------	----------

- So, we have a finite set of symbols  $\Sigma = \{A_1, A_2, \dots, A_n\}$ , and an axiom  
Then what?
- What was the cell split thing again?
  - In this case, it substitutes one symbol with itself two times.  
And in a more general case?

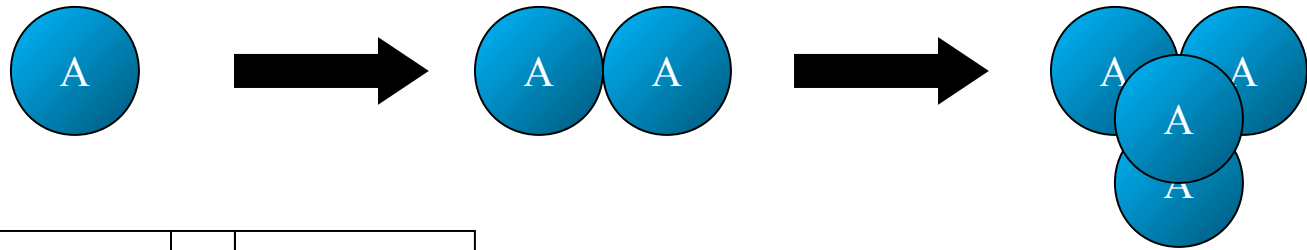
# Modeling the math



$\Sigma = \{A_1, A_2, \dots, A_n\}$	$\alpha$
-------------------------------------	----------

- So, we have a finite set of symbols  $\Sigma = \{A_1, A_2, \dots, A_n\}$ , and an axiom  $\alpha$ .  
Then what?
- What was the cell split thing again?
  - In this case, it substitutes one symbol with itself two times.  
And in a more general case?
  - Basically it substitutes to one symbol another one, or two of them, or three..
  - This we can model by saying that we substitute an element of  $\Sigma$  the juxtaposition of elements of  $\Sigma$ .
  - This is called a *transition rule*, or *rewriting rule*.
  - It associates to symbols  $\rightarrow$  sequences of symbols  
 $a_i \rightarrow \alpha_j$ , where  $\alpha_j \in \Sigma^*$ .

# Modeling the math



$\Sigma = \{A_1, A_2, \dots, A_n\}$	$\alpha$	$\Pi: \Sigma \rightarrow \Sigma^*$
-------------------------------------	----------	------------------------------------

- The transition rule set is called the set of production rules  
 $\Pi: \Sigma \rightarrow \Sigma^*$



# Modeling the math

$\Sigma = \{A_1, A_2, \dots, A_n\}$	$\alpha$	$\Pi: \Sigma \rightarrow \Sigma^*$
-------------------------------------	----------	------------------------------------

- One starts from the axiom
- Applies the production rules, step by step, and all at once, on the “current string”
- It generates new strings
  - These strings represent “the evolution of the cell”
- This abstraction mechanism is called a Lindenmayer system, from Lindenmayer, who invented this.

# Lindenmayer system (definition)

- A Lindenmayer system (*L-system*) is a system composed by
  - An alphabet  $\Sigma$ .
  - An axiom  $\alpha \in \Sigma^*$ .
  - A set of production (or *rewriting*) rules  $\Pi: \Sigma \rightarrow \Sigma^*$ .

$$L = \langle \Sigma, \alpha, \Pi \rangle$$

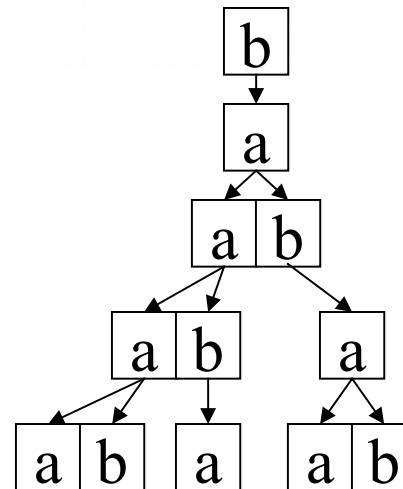
U

- The name Lindenmayer is taken from the inventor of Lindenmayer systems
- Note: Sometimes production rules such that a symbol of the alphabet is rewritten only as itself are listed separately

# Example

- Let the alphabet consist of the letters  $a, b$
- Suppose we have two production rules:
  - $a \rightarrow ab$
  - $b \rightarrow a$
- And suppose that the axiom is  $b$
- Then we obtain that we can generate the following strings

- $b$   
 $a$   
 $aba$   
 $abaab$   
 $abaababa$   
....
- Or, more figuratively:



# Interpreting L-systems

- The strings produced by L-systems are just strings
- To produce images from them one must interpret those strings geometrically
- There are two common ways of doing this
- Geometric replacement: each symbol of a string is replaced by a geometric element
  - Example: replace symbol X with a straight line and symbol Y with a V shape so that the top of the V slings with the end of the straight line
  - Example: XXYYXX

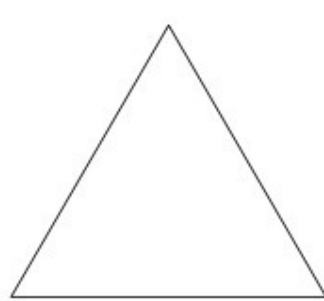


# Interpreting L-systems

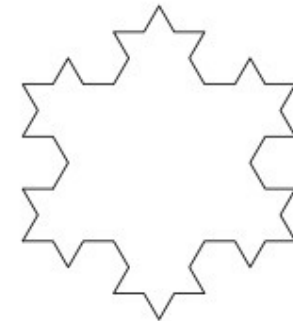
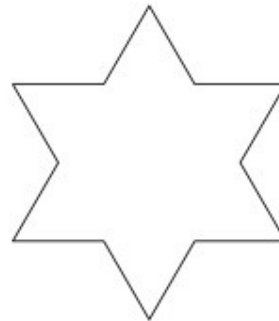
- Use turtle graphics: the symbols of the string are interpreted as drawing commands given to a simple cursor called *turtle*
- The state of a turtle at a given time is expressed as a triple  $(x, y, \alpha)$  where  $x, y$  give the coordinate of the turtle in the plane, and  $\alpha$  gives the direction of it is pointing to with respect to a given reference direction
- Two more parameters defined by the user are also used:
  - $d$ : linear step size
  - $\delta$ : rotational step size
- Given the reference direction, the initial state of the turtle  $(x_0, y_0, \alpha_0)$ , and the parameters  $d$  and  $\delta$  the user can generate the turtle interpretation of the string containing some symbols of the alphabet

# The Koch curve

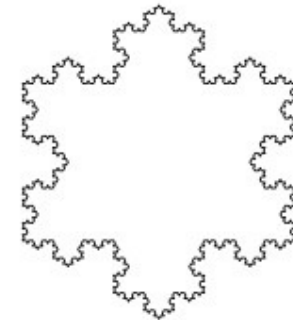
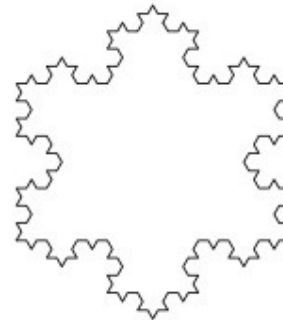
- The Koch curve is one of the most famous Lindenmayer generated systems.
- Fractal curve!



*initiator*



*generator*

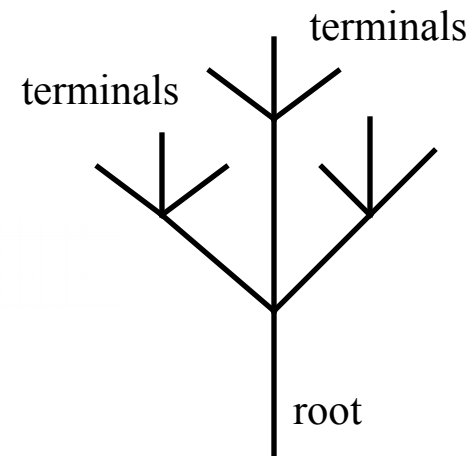


# D0L-systems

- Lindenmayer systems (or L-systems) such as the ones seen before
  - apply production rules independently from left and right strings surrounding the symbol
  - The production rule is thus applied free from the context the symbol is embedded into.
- Such systems are called *context free Lindenmayer systems*, or *D0L-systems*.
- Context free L-systems are deterministic:
  - From the same string one obtains always the same next generation strings

# Rooted trees

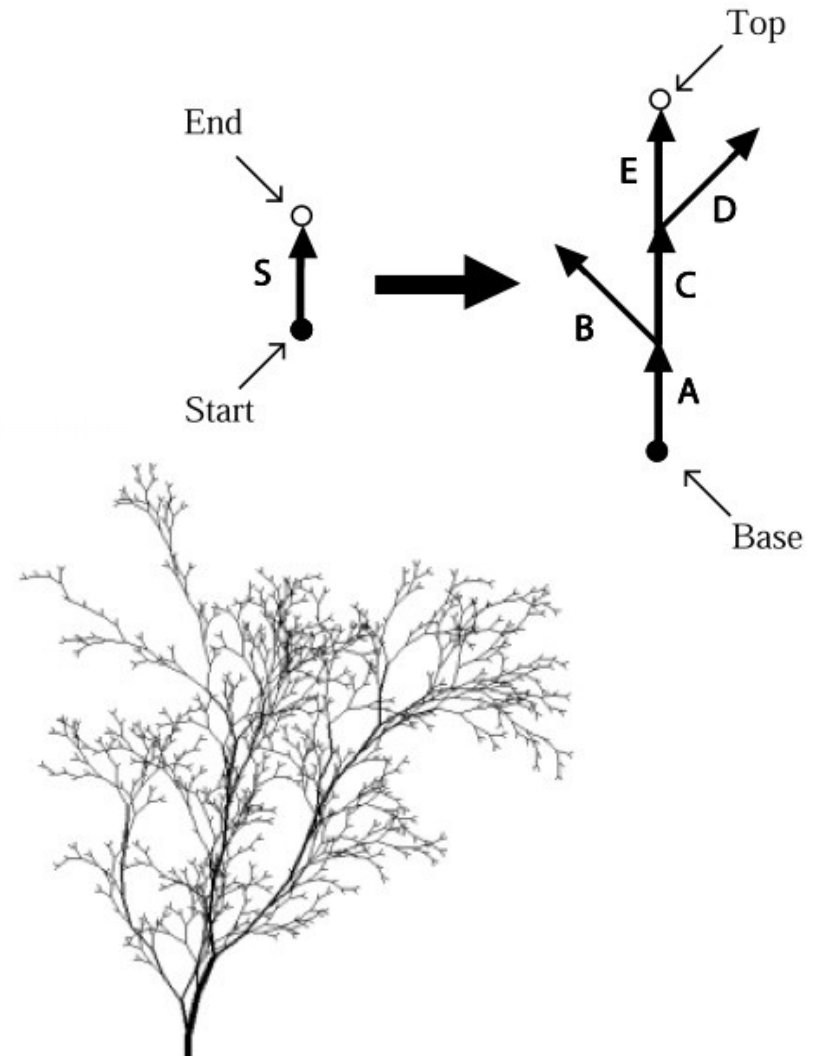
- Lindenmayer systems (or L-systems) such as the ones seen before are great to model branching structures, such as trees
- Among branching structures, *rooted trees* are used to model trees
- A rooted tree is a collection of edges which are labeled and directed
- A rooted tree has one particular edge: the *root*.
- All edges are similar to branch segments on a tree:
  - they connect to a father segments, and
    - either have themselves children segments, in which case they are called *intermediate* segments
    - Or have no children and are called *terminal* segments or *apex*



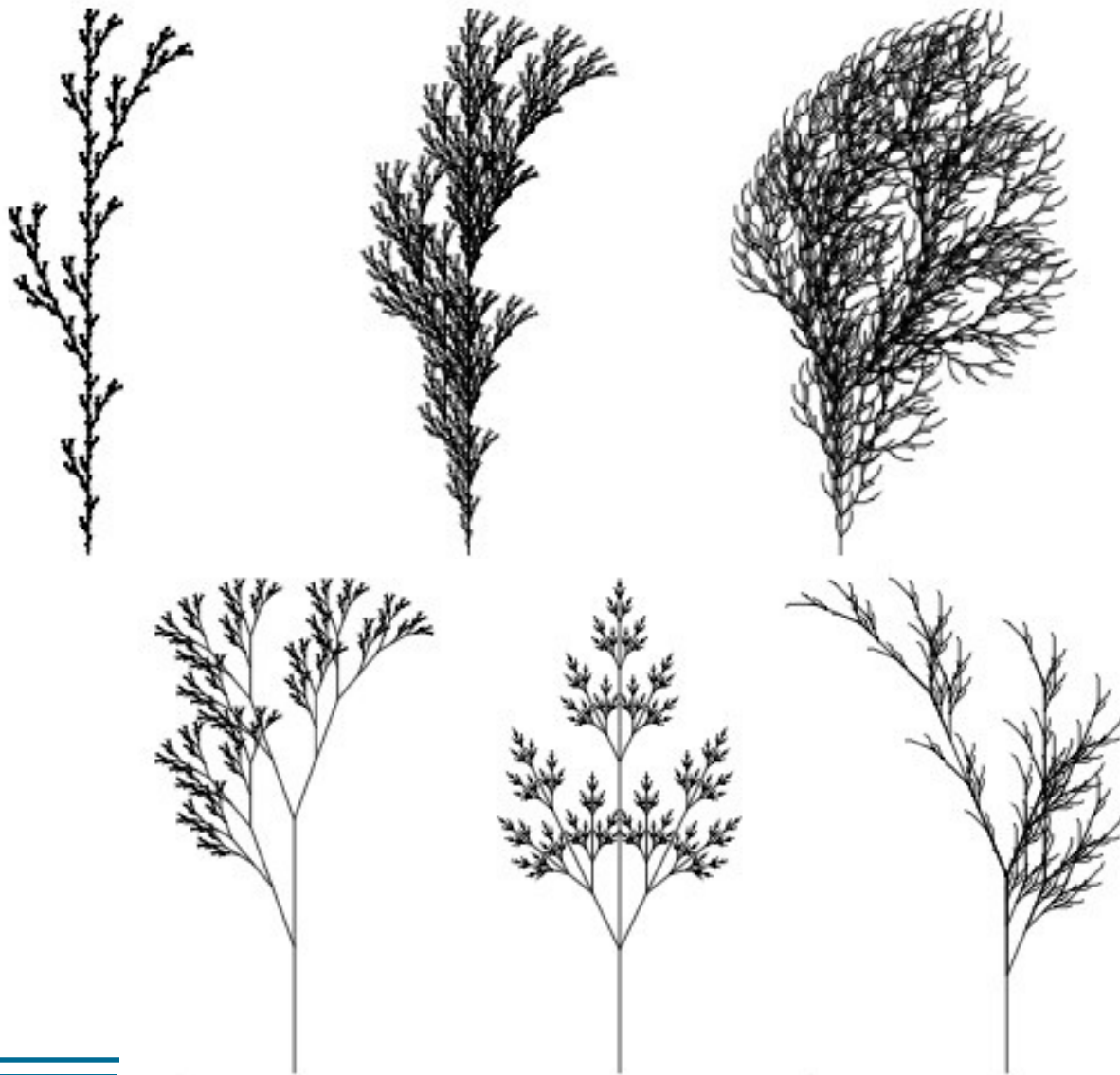


# Axial trees

- Among rooted trees there are axial trees:
  - For each intermediate node, at most one of its children is distinguished
  - The other children are called *side* or *lateral segments*
- An *axis* is a sequence of incident segments such that
  - It originates at the root or at a lateral branch
  - It ends with an apex
  - It is made of consecutive segments
- An axis, with its descendants is called a *branch*.
- On trees, productions “look” much more complicated, but they are not: simple substitutions
- If productions are context free, we speak of a tree D0L-system



# Some examples



# Stochastic L-systems

- Context free L-systems are deterministic:
  - given the same seed, they reproduce always the same tree
- Is this realistic?
  - Not much




# Stochastic L-systems

- Context free L-systems are deterministic:
  - given the same seed, they reproduce always the same tree
- Is this realistic?
  - Not much
  - If nature would handle like this, we would have all trees looking the same.
  - This is why scientists have added a small but relevant modification to L-systems.....

# Stochastic L-systems

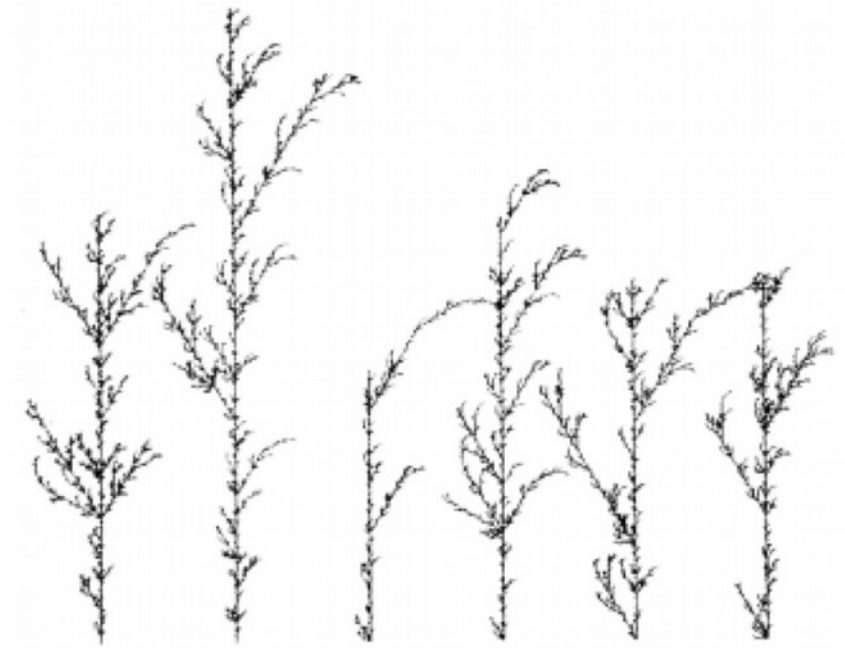
- Context free L-systems are deterministic:
  - given the same seed, they reproduce always the same tree
- Is this realistic?
  - Not much
  - If nature would handle like this, we would have all trees looking the same.
  - This is why scientists have added a small but relevant modification to L-systems.....
  - ...by adding **randomness** to the transition function!

# Stochastic L-systems

- And how do I add randomness to the transition function?
- Remember, an L-system  $L = \langle \Sigma, \alpha, \Pi \rangle$  was a triplet of
  - An alphabet  $\Sigma$ .
  - An axiom  $\alpha \in \Sigma^*$ .
  - A set of production (or  *rewriting*) rules  $\Pi: \Sigma \rightarrow \Sigma^*$ .
- Instead of fixed rewriting rules, we need to add a probability function
$$P: \Pi \rightarrow [0, 1]$$
that associates to the production rules a probability of being chosen.
- For a given symbol A of the alphabet we will have therefore a list of production rules
$$A \rightarrow \alpha_0 \text{ (prob. } p_0)$$
$$A \rightarrow \alpha_1 \text{ (prob. } p_1)$$
$$\dots$$
$$A \rightarrow \alpha_h \text{ (prob. } p_h),$$
whereby the sum of these probabilities equal 1.
- Example:
$$A \rightarrow ABA \text{ (prob. 25\%)}$$
$$A \rightarrow ABAA \text{ (prob. 75\%)}$$
- Any time we have to apply a production rule to the letter of the alphabet, we draw a random number in  $[0, 1]$ .
- If it is smaller than 0.25 we choose the 1st production, otherwise the 2nd.
- The new productions are called *stochastic productions*

# Stochastic L-systems

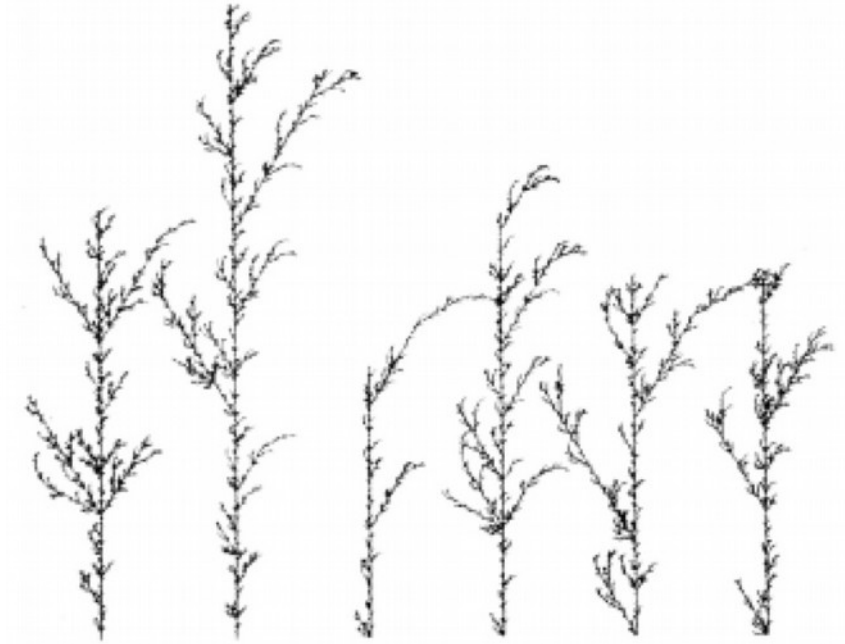
- This way from the same axiom one can derive different branching shapes..
- ...and more realistic pictures





# Stochastic L-systems

- This way from the same axiom one can derive different branching shapes..
- ...and more realistic pictures



- It looks good, but does not model what nature does!

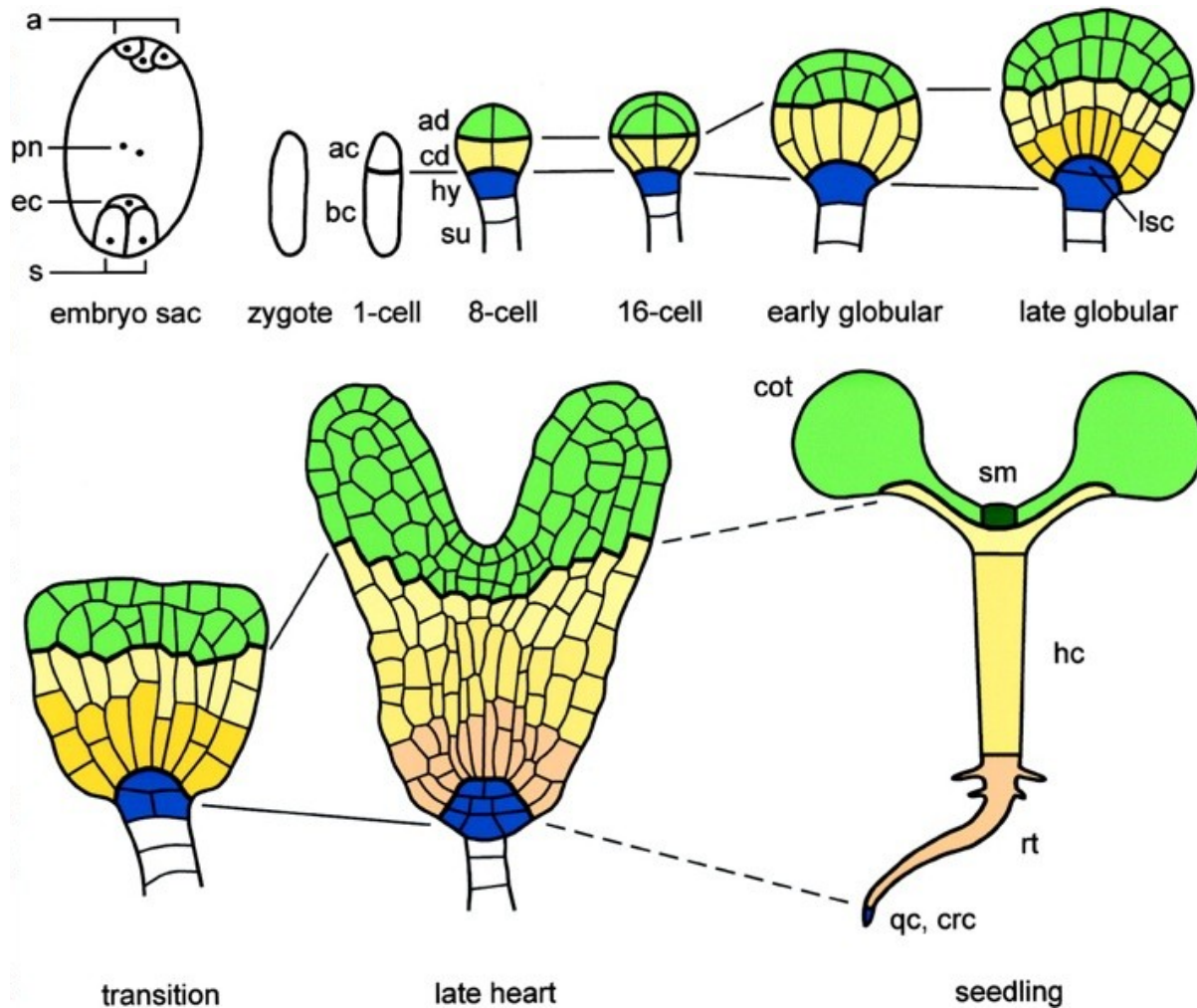


# A further example

- Let us get back to the example of the last time....
- For a long time, botanists have been trying to describe and understand how plants develop
- For a long time, the plant called *arabidopsis thaliana* has been used as a good observation object for plant growth
- There are plenty of observations of how this plant develops from a cell

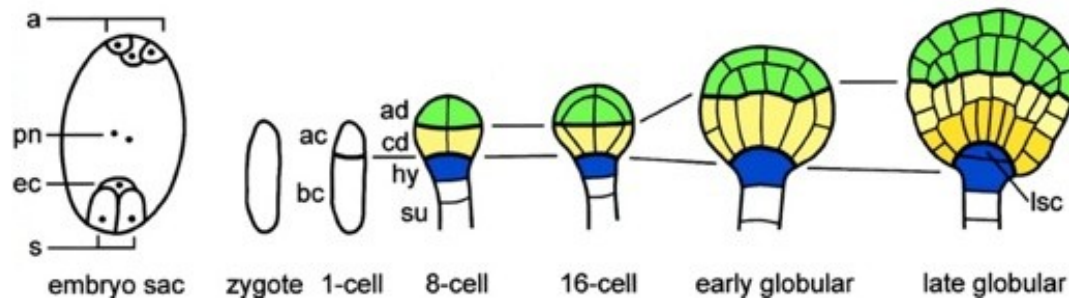


# A further example



First phase of growth *arabidopsis thaliana*

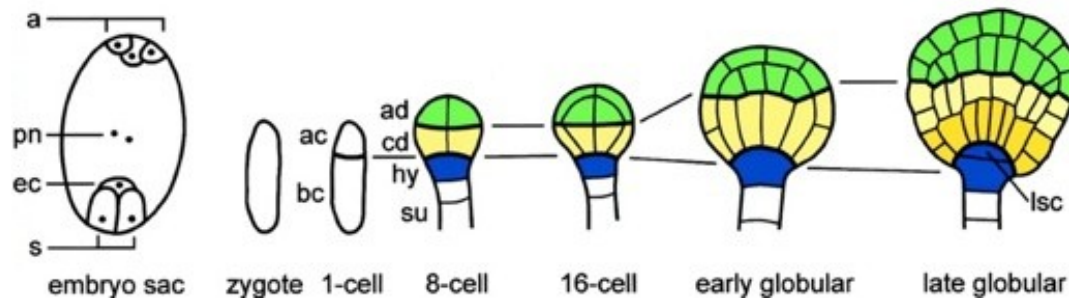
# Reasoning on the second example



- What can one learn from this picture?

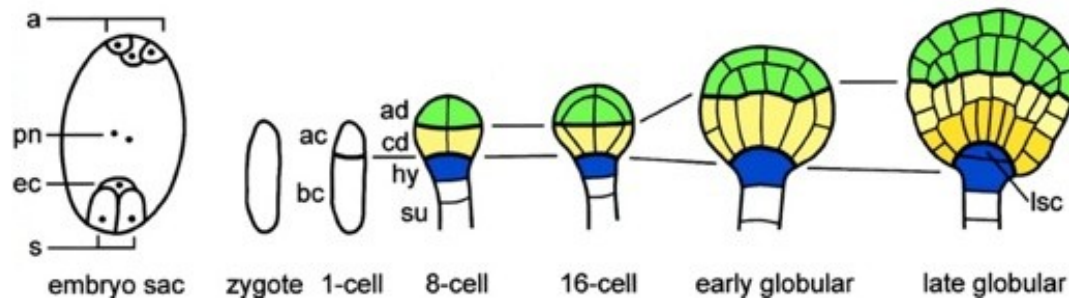
U

# Reasoning on the second example



- What can one learn from this picture?
  - Cells differentiate in time
  - Start growing in different ways
    - depending on their position
    - and on their surroundings
  - Unfortunately our cell development model is too rigid: a cell develops always in the same way.
  - If the production rules in  $\Pi$  are such that the left hand side is a single element, the Lindenmayer system is called a *DOL-system*, or *context free Lindenmayer system*
  - BUT one can learn from this!

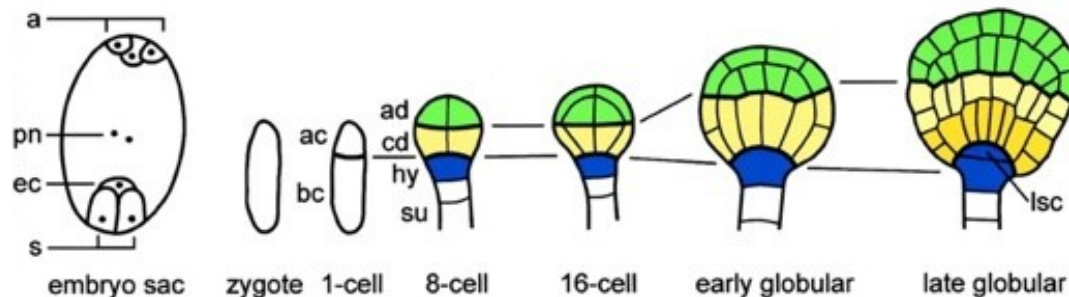
# Reasoning on the second example



- Remember: we had for D0L-systems that the productions were  

$$\Pi: \Sigma \rightarrow \Sigma^*$$
 associating  $a_i \rightarrow \alpha_j$
- Who says that at the left hand side of the productions we need to have a single element?
- Can't it be that depending on what the symbol to which we apply the production has around we can have two different production rules applied?
- If  $a_i$  is preceded by a certain string, and followed by a certain other string, we apply one production rule.
- If instead it is preceded and followed by other strings we apply another rule.

# Reasoning on the second example



- In this case we have that the productions are still defined similarly

$$\Pi: \Sigma \rightarrow \Sigma^*$$

but they associate to  $a_i$

$$\alpha_r a_i \alpha_s \rightarrow \alpha_j$$

in case that  $a_i$  is preceded by  $\alpha_r$  and followed by  $\alpha_s$ , while if it is preceded and followed by two different strings  $\alpha_p$  and  $\alpha_q$ , with

$\alpha_p \neq \alpha_r$  and  $\alpha_q \neq \alpha_s$ , then a different production rule is applied:

$$\alpha_p a_i \alpha_q \rightarrow \alpha_m$$

with  $a_m \neq a_i$ .

- This means that AAB**C** gets transformed into AAD**C**, but if we had CC**B**A we get CC**E**A
- L-systems such that they use such production rules are called *D2L-systems*, whereby the 2L indicates that both sides of the letter to be substituted influence the production to be applied.
- If instead the production rule is influenced only by one side, they are called *D1L-systems*.
- D1L- and D2L-systems are said to be *context sensitive systems*.

# Context-sensitive L-systems

- Let us recap the definition:
- A *D2L-system* is a triplet  $L = \langle \Sigma, \alpha, \Pi \rangle$ , where
  - $\Sigma$  is an alphabet,  $\Sigma^*$  is the set of the juxtaposition of letters of  $\Sigma$ .
  - $\alpha \in \Sigma^*$  is an axiom
  - The context sensitive set of production rules applied to letters of the alphabet is

$$\begin{aligned} \Pi: \quad \Sigma &\longrightarrow \Sigma^* \\ \alpha_r a_i \alpha_s &\longrightarrow \alpha_j \\ \alpha_p a_i \alpha_q &\longrightarrow \alpha_m \end{aligned}$$



# Context-sensitive L-systems





# Context-sensitive L-systems

- Context sensitive L-systems model better nature, since you can make the production rules dependent on the context you immerse your L-system into.
- For example, trees in a forest develop new branches only at the tip, so as to maximize light captured...



# Context-sensitive L-systems

- Context sensitive L-systems model better nature, since you can make the production rules dependent on the context you immerse your L-system into.
- ...or external forces, such as wind, force trees to develop opposite of the direction of prevailing wind...



# Context-sensitive L-systems

- Context sensitive L-systems model better nature, since you can make the production rules dependent on the context you immerse your L-system into.
- ..or trees moving away from the façade of a house to collect more light. U





# Context-sensitive L-systems

- Context sensitive L-systems model better nature, since you can make the production rules dependent on the context you immerse your L-system into.
- ..or trees moving away from the façade of a house to collect more light. U
- All these factors can be incorporated into the production rules to model nature.



# Parametric and timed L-systems

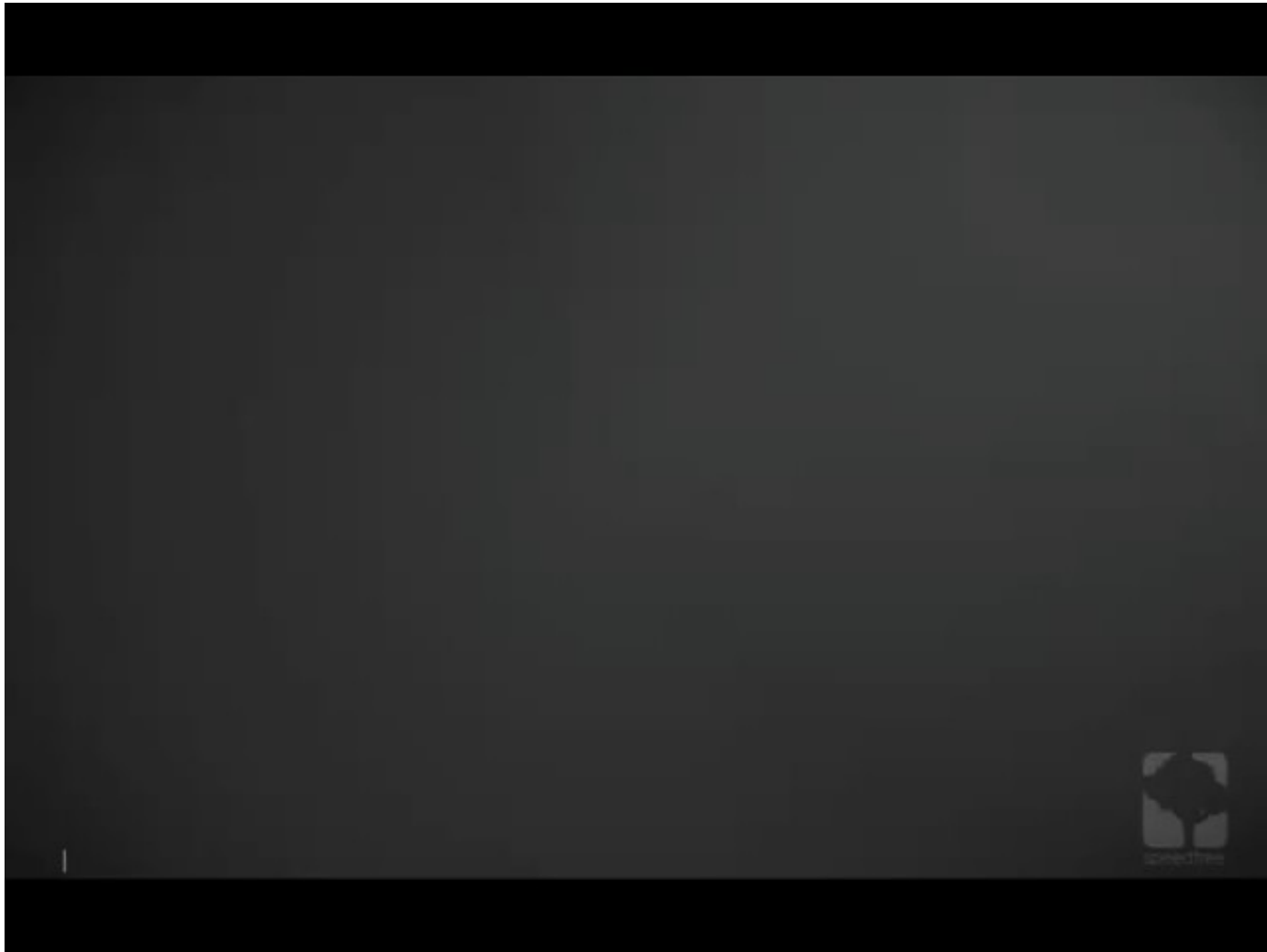
- In parametric L-systems, symbols can have one or more parameters associated to them
- These parameters can be set and modified by the productions of the L-system
- Additionally, optional conditional terms can be associated with the productions
- All this to simulate differences in the change through time in a plant



# Parametric and timed L-systems

- Timed L-systems add two things
  - A global time variable helping control the evolution of a string And a local age value  $\tau_i$  assoc. with each letter  $\mu_i$ .
  - The production
$$(\mu_0, \beta_0) \rightarrow ((\mu_1, \alpha_1), \dots, (\mu_n, \alpha_n))$$
indicates that  $\mu_0$  has a terminal age of  $\beta_0$ .
  - Each symbol has one and only one terminal age
  - When a new symbol is generated, it is initialized at age 0 and exist until it reaches
  - After its lifespan ends, the symbol will become something else and „mutate“

# Let's look at where we are!



# Forests

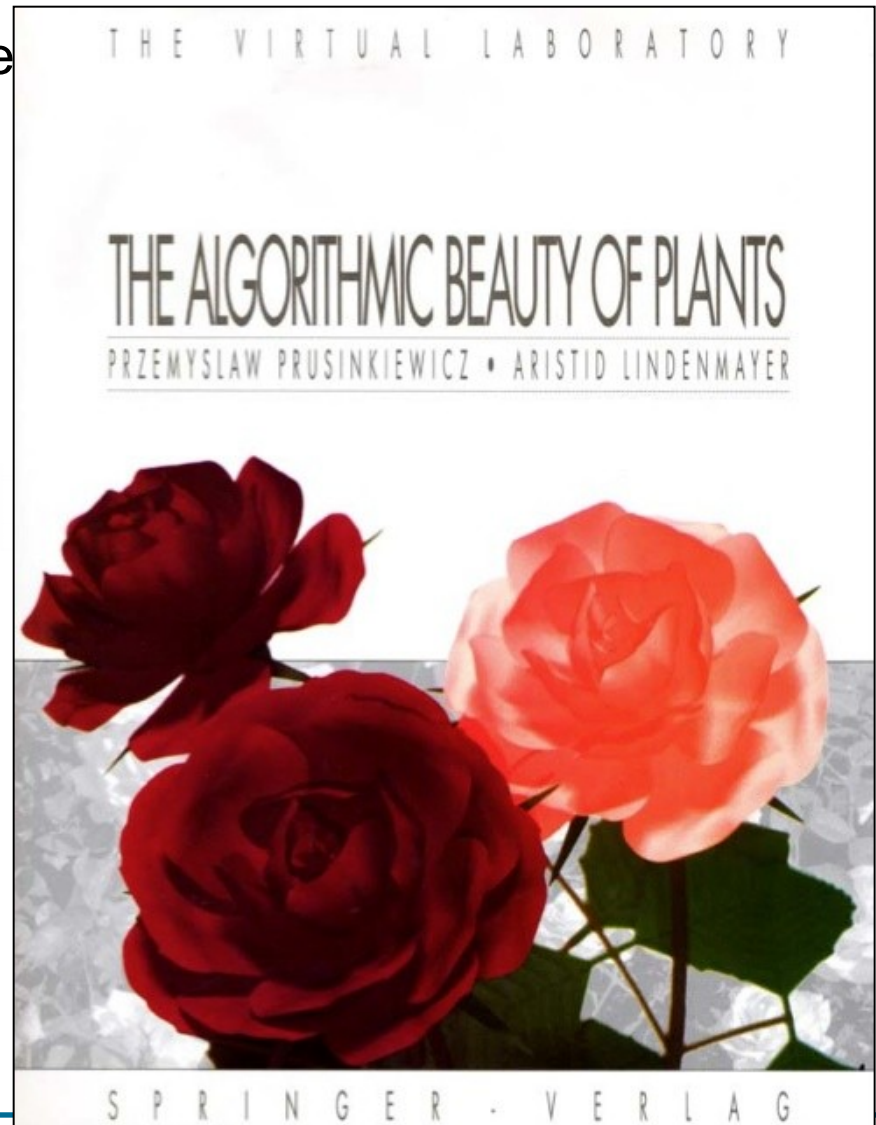
- Do the trees of a forest have to look similar to each other?
- What about forest borders?
- How do I render the forest?
- Close view?
- Far view?
- Wind?





# More information

- Read the literature, if you are interested!



# Water

- Water is challenging: its appearance and motion take various forms
- Modeling water can be done by adding a bump map on a plane surface
- Alternatively, one can use a rolling height field, to which ripples are added later in a postprocessing step
- When doing ocean waves, water is assumed not to get transported, although waves do travel either like sinus or cicloiddally
- If water has to be transported (=flow) this adds a lot of computational complexity



# Small waves

- Simple way: big blue polygon
- Add normal perturbation with sinuisoidal function and you have small waves
- Usually you would start sinuisoidal perturbation from a single point called source point
- Sinus perturbation has, however crests of the same amplitude. This is not so realistic, and waves can be perturbed through smaller radial waves to achieve non self-similarity
- Similarly, one can superimpose more different sinuisoidal waves to achieve an interesting complex surface
- All these methods give a first decent approximation, but not always very realistic

# Wave functioning

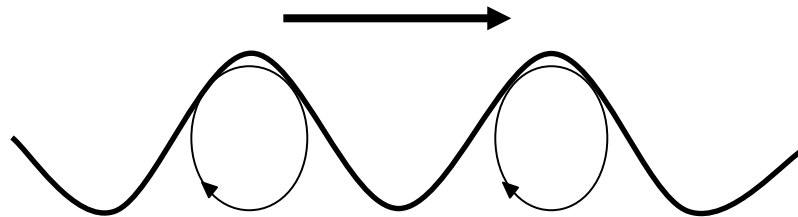
- A better way of doing water is to incorporate physical laws
- There is a variety of types of waves:
  - Tidal waves
  - Waves created by the wind
- In general, at a distance  $s$  of the sourcepoint we have that

$$f(s, t) = A \cos\left(\frac{2\pi(s - Ct)}{L}\right)$$

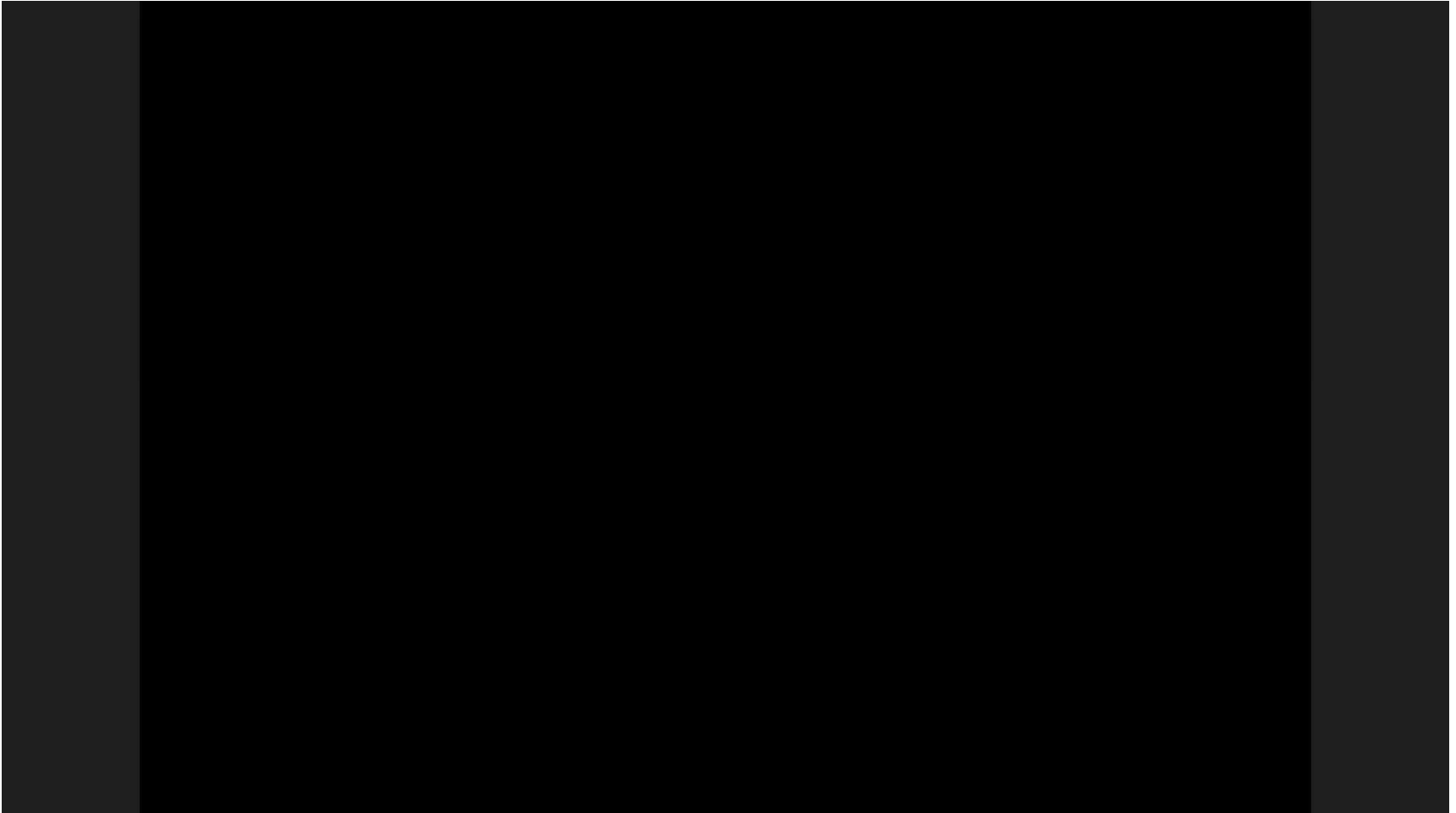
- Where
  - $A$  maximum amplitude
  - $C$  speed of propagation
  - $L$  wavelength  
(it holds  $C=L/T$ , with  $T$  time for one wave cycle to pass a given point (freq.))
  - $t$  time
- Waves move differently from the water itself. A water particle would almost move circularly:
  - Follow wave crest, sink down and move backwards, then come up again

# Wave functioning

- Small waves (with little steepness) work almost like sinus curves
- The bigger they get, the more they look like a sharply crested peak, i.e. They approach the shape of a cycloid (point on wheel)
- When a wave approaches the shoreline, at an angle, the nearest part to the coastline slows down
- While its speed  $C$  and wavelength  $L$  reduce near the coast, its period stays the same and amplitude remains the same or increases.
- But because the speed of the water particles remains the same, the wave tends to break as it approaches the shore
- Literally, particles are „thrown forward“ beyond the front of the wave



# Wave example



Youtube: Eric Goodwin

# Wave example



Copyright © 1986 A.Founier, W.T. Reeves

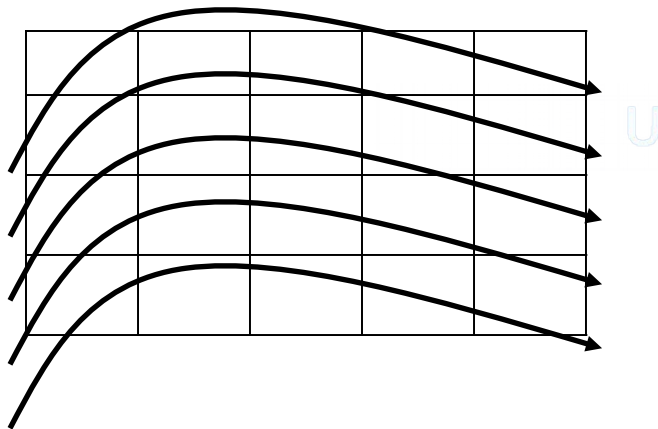
# Gaseous Phenomena

- Gas is quite complicated to do
- But occurs often (smoke, fire, clouds)
- Fluid dynamics long studied, and applies to both gas and liquids
  - Incompressible --> Liquid
  - Compressible --> Gas
- There are different types of movement in fluids
  - Steady state flow: velocity and acceleration at any point in space are constant
  - Vortices: circular swirls of material,
    - depend on space and not on time in steady state flow
    - In time varying flow, particles carrying non zero vortex strength travel through the environment and „push“ other particles. This can be simulated by using a distance-based force

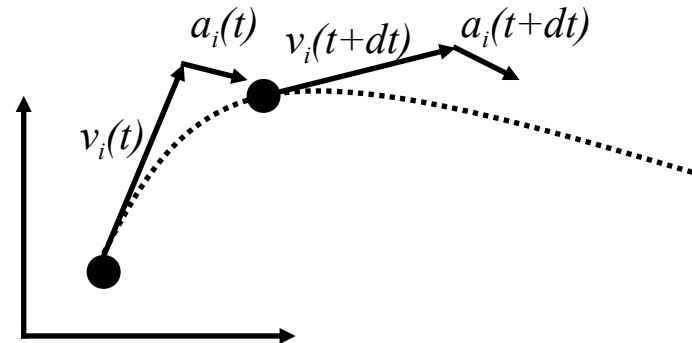


# Gaseous phenomena

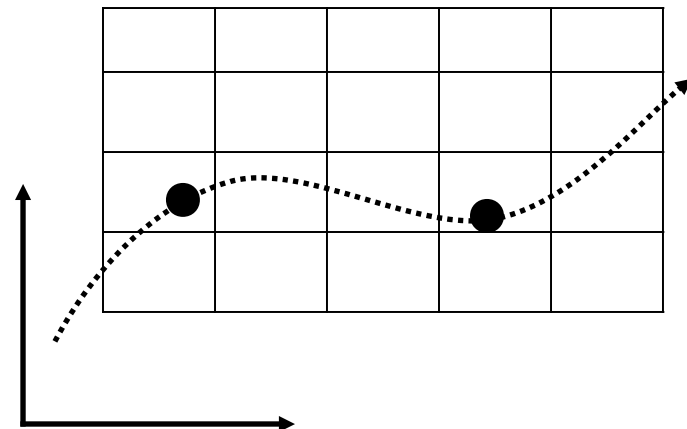
- There are 3 main approaches to modeling gas:
  - Grid-based methods (Eulerian formulation)



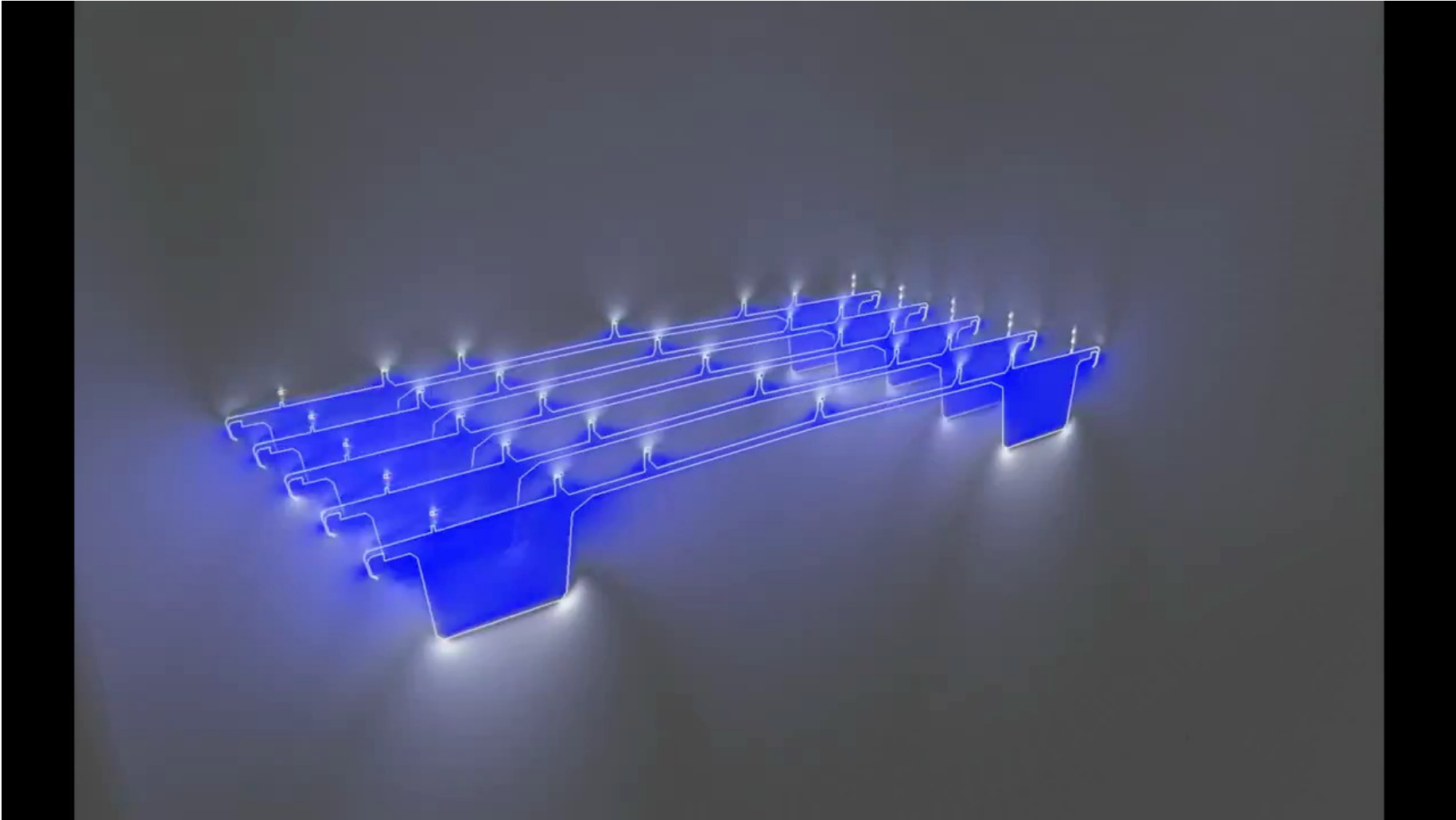
- Particle-based methods (Lagrangian formulation)



- Hybrid methods

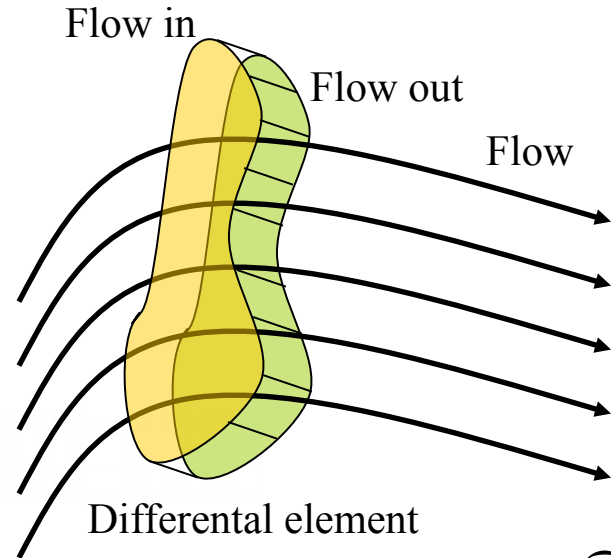


# Gaseous phenomena



# Computational fluid dynamics

- CFD solves the physical equations directly
- Equations are derived from the Navier-Stokes equations
- Standard approach is based in a grid: set up differential equations based on conservation of momentum, mass and energy in and out of differential elements
- Quite complicated



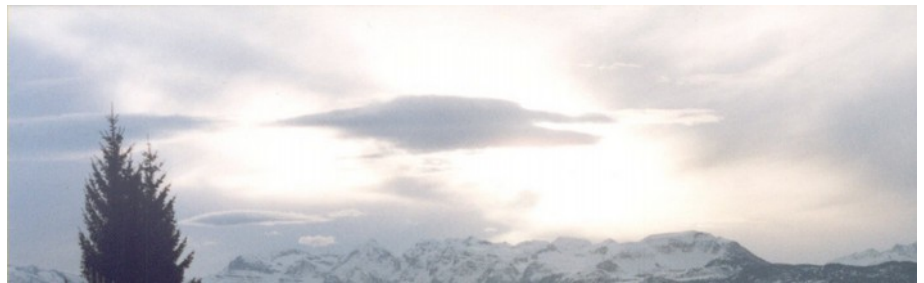
Courtesy Japan Aerospace  
Exploration Agency (JAXA)

# Clouds

- The biggest problem with clouds is that we are so familiar with them, i.e. we know well realistic looking ones
- Made of ice crystals or water droplets suspended on air (depending on temperature).
- Formed when air rises, and humidity condensates at lower temperatures
- Many many shapes: cirrus, stratocumulus, cumulus

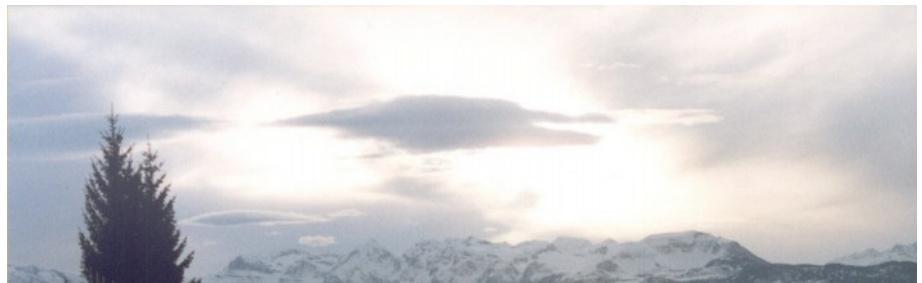


Courtesy Daniel Bramer, UIUC



# Clouds

- Clouds have different detail at different scales
- Clouds form in a turbulent chaotic way and this shows in their structure
- Illumination characteristics are not easy, and vary because the ice and water droplets absorb, scatter and reflect light
- There are two illumination model types for clouds:
  - low albedo
  - High albedo



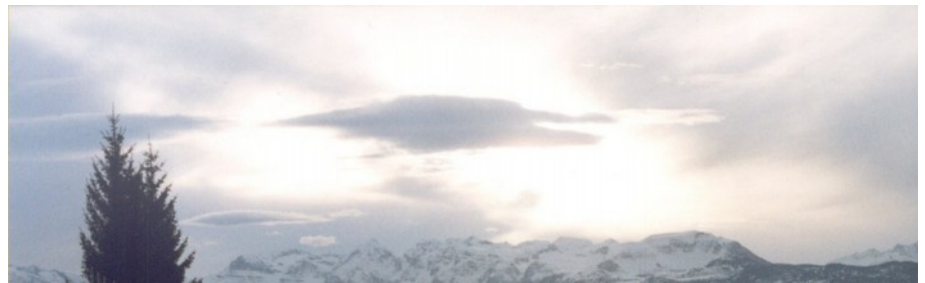
Courtesy Daniel Bramer, UIUC

# Cloud illumination

- Low albedo: assumes that secondary scattering effects are negligible
- High albedo: computes secondary order and high order scattering effects
- Optically thick clouds like cumuli need high albedo models
- Self shadowing and cloud shadowing on landscape have also to be considered



Courtesy Daniel Bramer, UIUC





# Cloud illumination: surface methods

- Early models used either by using Fourier synthesis to control the transparency of large hollow ellipsoids
- Others used randomized overlapping spheres to generate the shape
- A solid cloud texture is combined with transparency to control the transparency of the spheres
- Transparency near the edges is increased to avoid seeing the shape of the spheres
- Such surface models are not so realistic, because the surfaces are hollow

# Cloud illumination: volume methods

- More accurate models have to be used in order to capture the 3D structure of a cloud [Kajiya, Stam and Fiume, Foster and Metaxas, Neyret]
- Meyret did a model based of a convective cloud model using bubbling and convection precesses
- However, it uses large particles (surfaces) to model the cloud structure
- One can use particle systems, but a very large number of particles is needed
- Other approaches use volume-rendered implicit functions, sometimes combining them with particle systems approaches
- Implicit functions rendering can be used on the large scale, to define the global structure of a cloud, and combined with simpler procedural techniques to produce the detail
- To add a „bit“ to complexity, clouds also need to be animated since they change in time



# Cloud example



Courtesy Andrew Price, [blenderguru.com](http://blenderguru.com)

# Fire

- Fire is even more difficult:
  - it has the same complexity of gas and clouds
  - but has very violent internal processes producing light and motion
- Recently, good advances were made
- At the „exactness“ limit of the models, CFD can be used to produce fire and simulate its internal development, but it is difficult to control
- Studies on simulating the development and spreading of fire began to appear, but are usually not concerned with the internal processes within fire.

# Fire: particle systems

- Computer generated fire has been used in movies since a long time, exactly since Star Wars II
- In this film, an expanding wall of fire spread out from a single impact point
- The model uses a two-level hierarchy of particles
  - First level at impact point to simulate initial ignition
  - Second level: concentric rings of particles, timed to progress concentrically to form a wall of fire and of explosions
- Each of these rings is made of a number of particle systems positioned on the ring and overlapping with neighbors so as to form a continuous ring.
- The individual particle systems are modelled to look like explosions
- Particles are oriented to fly up and away from the planet surface
- The initial position of a particle is randomly chosen from the circular base of the particle systems
- Initial ejection direction is forced into a certain cone

# Fire: particle systems



<http://www.miikahweb.com/en/articles/blenderfire>

# Fire: other approaches

- Two dimensional animated texture maps have been used to simulate a gas flame
- This works however only in one direction
- Others (Stam and Fiume) presented advection-diffusion equations to evolve both density and temperature fields
- The users control the simulation by specifying the wind field

U



Copyright (c) 1988 ILM

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++