Copyright © by Wojtek Sterna

Math for 3D/Games Programmers

8. Derivatives

Copyright © by Wojtek Sterna

Table of Contents

- 8.1. What is a Derivative
- 8.2. Numerical Calculation
- 8.3. Analytical Calculation
- 8.4. Function Extrema
- 8.5. Derivative of Parametric Function
- 8.6. Partial Derivatives and Gradient
- 8.7. Derivative of Implicit Function
- 8.8. Least Squares Method
- 8.9. Gradient Descent Algorithm
- 8.10. Tangent Space

Copyright © by Wojtek Sterna

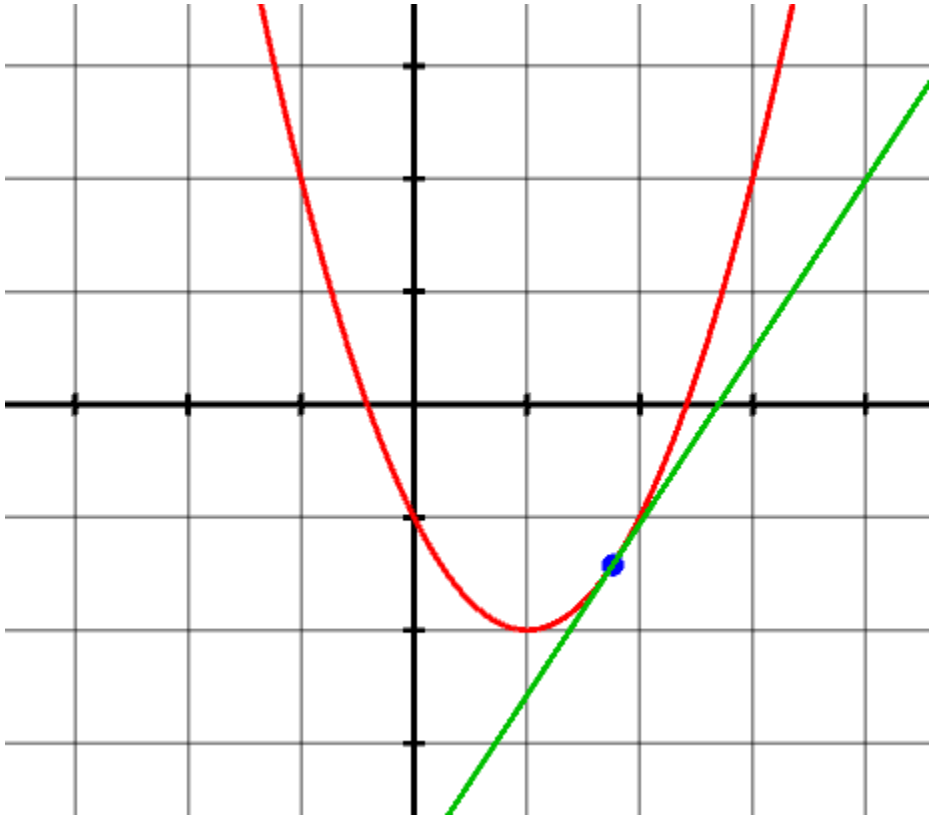
8.1. What is a Derivative

- **Derivative** is a measure of change of the value of function $f(x)$ with respect to change of x
- The derivative of function $f(x)$ is another function which we denote $f'(x)$
- Derivative can be calculated at any point of a function (provided that certain conditions are met)
- The process of calculating the derivative is called **differentiation**
- Graphically the derivative at x is equal to the slope a of the straight line that is tangent to $f(x)$ at x
- A particularly interesting case occurs when $f'(x) = 0$

Copyright © by Wojtek Sterna

8.1. What is a Derivative

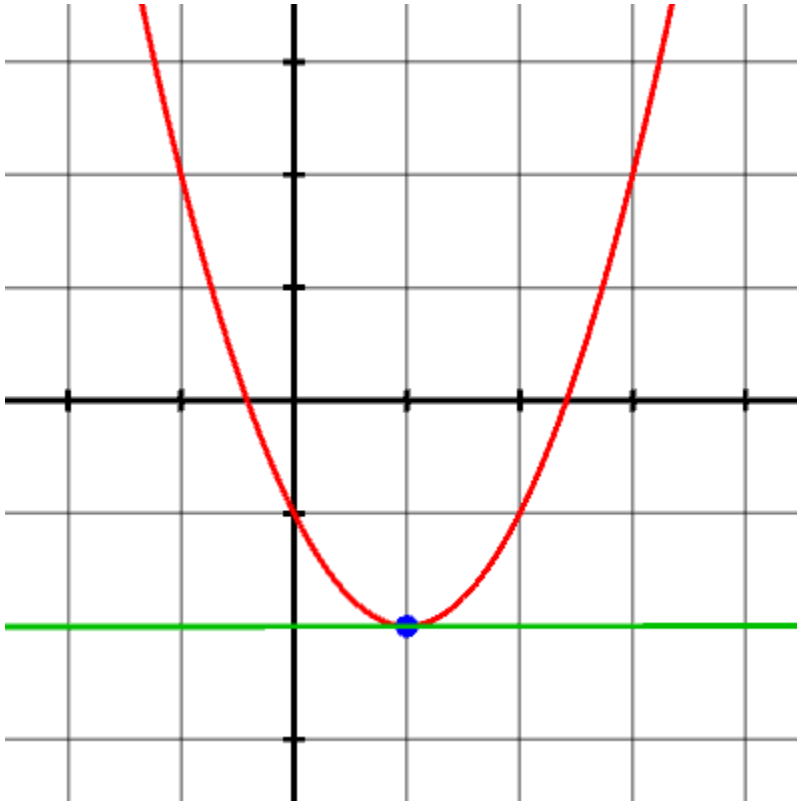
- Tangent to a function; $a = f'(x)$:



Copyright © by Wojtek Sterna

8.1. What is a Derivative

- Tangent with the slope $a = f'(x) = 0$:



Copyright © by Wojtek Sterna

8.1. What is a Derivative

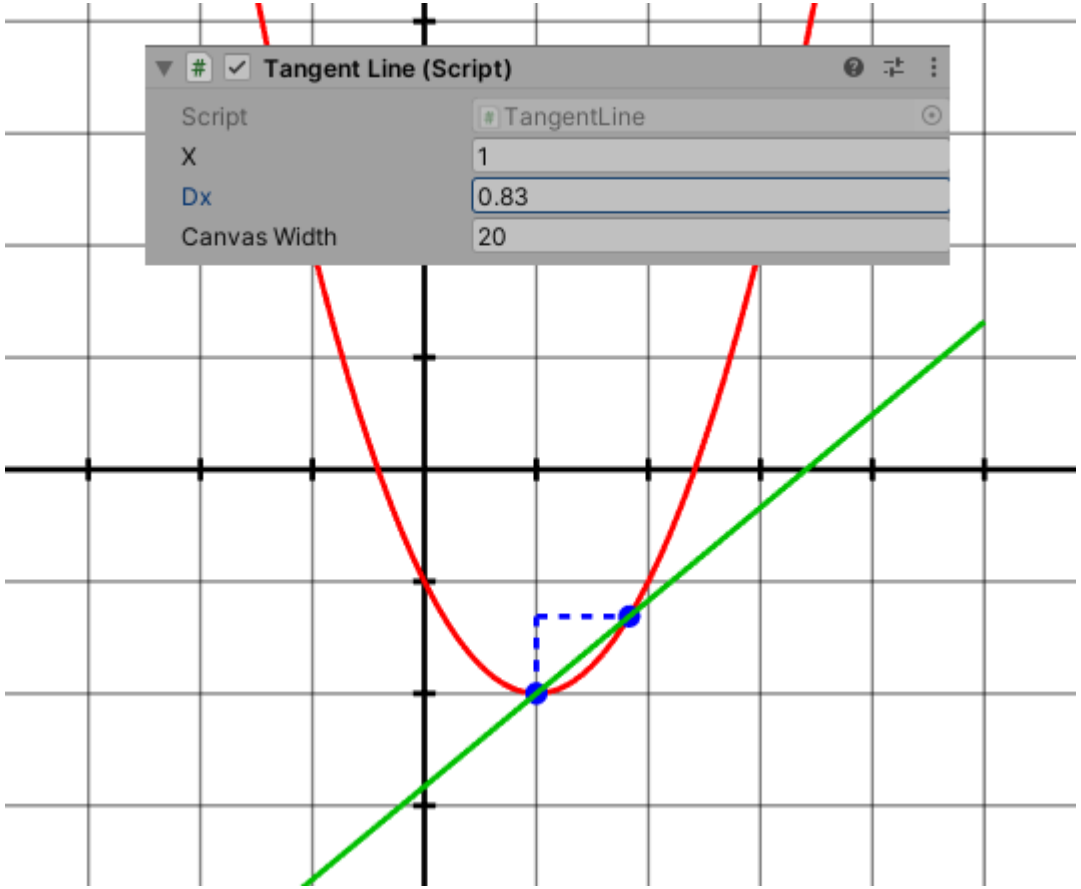
- If we take two points **that are close to one another** we can calculate the derivative at those points by calculating the slope a of the straight line that passes through those two points
- For two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ the straight line equation can be calculated with the following formulas:

$$\begin{cases} a = \dfrac{y_2 - y_1}{x_2 - x_1} \\ b = y_1 - ax_1 \end{cases}$$

- $f'(x_1) = f'(x_2) = a$, when x_2 is „infinitely close to" x_1

Copyright © by Wojtek Sterna

8.1. What is a Derivative

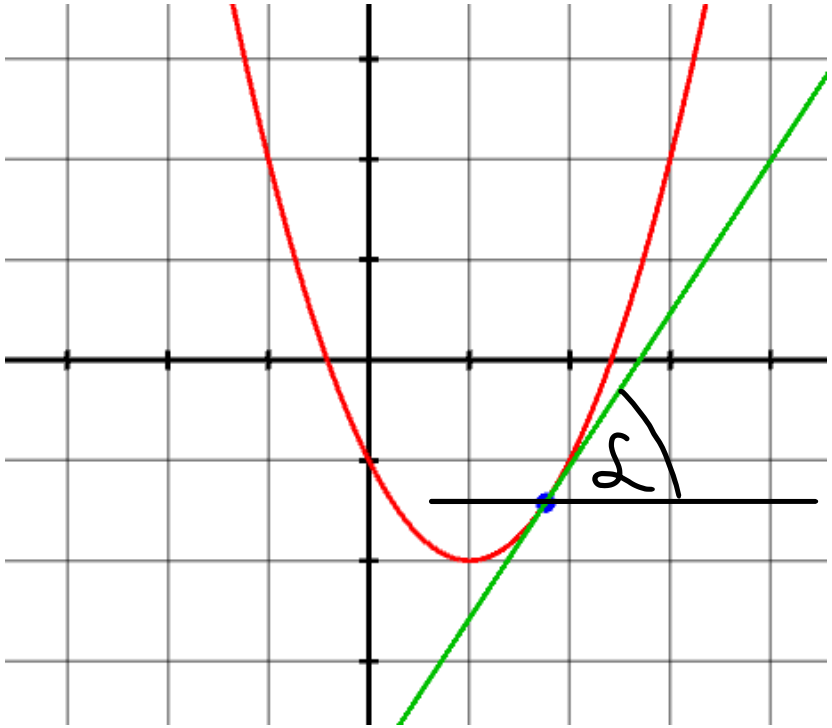


Copyright © by Wojtek Sterna

8.1. What is a Derivative

- For a straight line with slope a it is true that:

$$a = \tan(\alpha) \qquad \alpha - \text{angle between the line and the X axis}$$



Copyright © by Wojtek Sterna

8.1. What is a Derivative

- Usually the derivative of $f(x)$ is denoted as $f'(x)$, but there are also alternative notations:

$$\frac{df}{dx}$$

$$\frac{d}{dx}f(x)$$

$$\dot{f}$$

Copyright © by Wojtek Sterna

8.1. What is a Derivative

- To sum up: the derivative of $f(x)$ at x can be identified with the slope a of the straight line tangent to function f at x

Copyright © by Wojtek Sterna

8.2. Numerical Calculation

- Formal definition of the derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- By choosing smaller and smaller values of h we get more and more accurate value of the derivative
- Denote $x_2 = x + h$ and $x_1 = x$.
  Any associations?

Copyright © by Wojtek Sterna

8.2. Numerical Calculation

- $x_2 = x + h$ and $x_1 = x$:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f'(x) = \lim_{h \to 0} \frac{f(x_2) - f(x_1)}{h} \qquad h = x_2 - x_1$$

$$f'(x) = \lim_{h \to 0} \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$f'(x) = \lim_{h \to 0} \frac{y_2 - y_1}{x_2 - x_1}$$

Copyright © by Wojtek Sterna

8.2. Numerical Calculation

- Let's take $f(x) = x^2$, $x = 4$ and $h = 0.01$:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \qquad h > 0$$

$$f'(4) \approx \frac{(4+0.01)^2 - (4)^2}{0.01} = \frac{0.0801}{0.01} = 8.01$$

$$\text{for } h = 0.001 \quad \rightarrow \quad f'(4) = 8.001$$

Copyright © by Wojtek Sterna

8.2. Numerical Calculation

- Eventually, we can calculate numerically the derivative of any function using the formula:

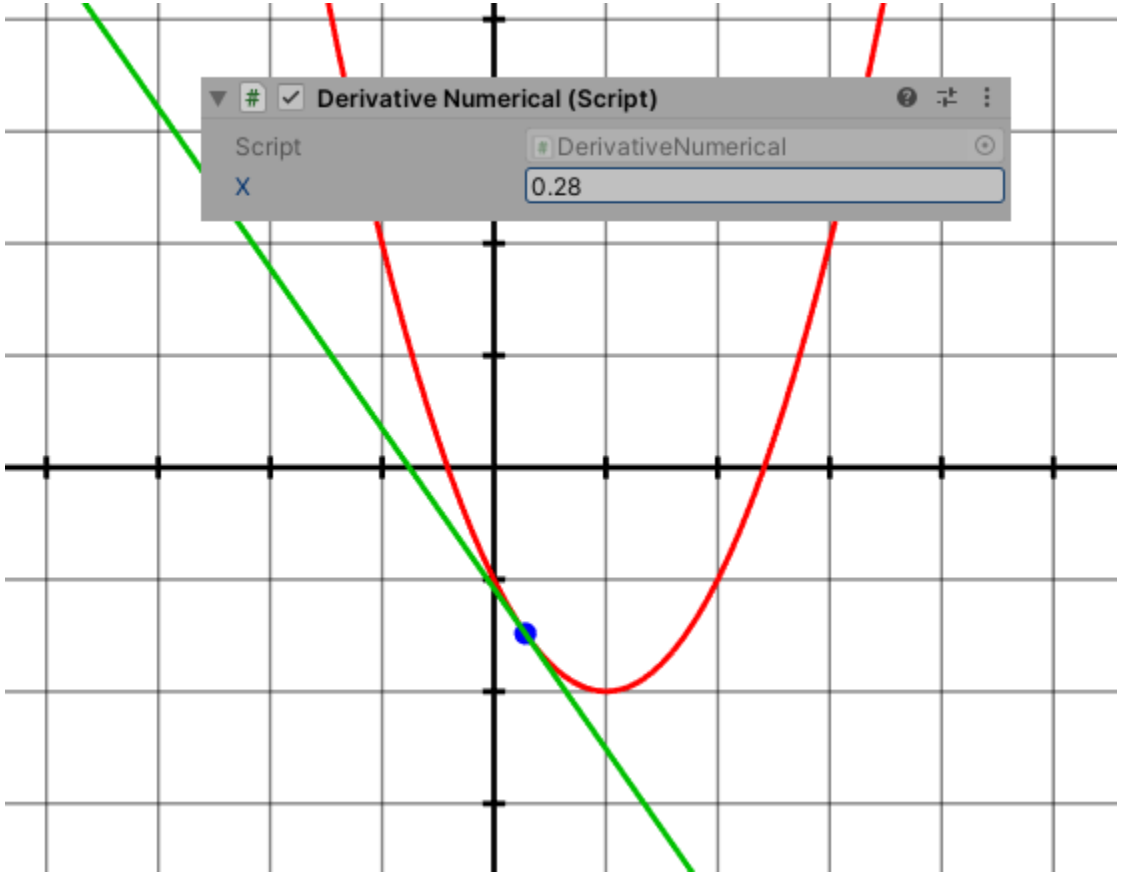
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

- This is the most basic formula; there are others
- Another one with a slightly better accuracy:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Copyright © by Wojtek Sterna

8.2. Numerical Calculation



Copyright © by Wojtek Sterna

8.2. Numerical Calculation

- A great advantage is the ease of calculating the derivative of any function
- It is possible to calculate the derivative of a function that is defined using **tabular data** (presented in some array, for example). Analytically it is very difficult
- A disadvantage is precision. The lower the parameter h the more accurate the derivative. In theory, because in practice „on the other side/end” there is a problem with the precision of calculations. Somewhere is the middle ground

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- If a function is represented with a formula we can calculate the derivative analytically
- Derivatives of elementary functions can quite easily be derived from the definition of the derivative
- Derivatives of compositions of functions and derivatives of the sum/difference/product/quotient of two functions are calculated using dedicated formulas. Those formulas can also be derived from the definition of the derivative

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- For $f(x) = x^2$:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \to 0} \frac{(x+h)^2 - (x)^2}{h}$$

$$f'(x) = \lim_{h \to 0} \frac{x^2 + h^2 + 2xh - x^2}{h}$$

$$f'(x) = \lim_{h \to 0} \frac{h(h+2x)}{h} = \lim_{h \to 0} (h + 2x) = 2x$$

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- Derivatives of a few elementary functions:

$$f(x) = cx \quad \rightarrow \quad f'(x) = c$$

$$f(x) = x^2 \quad \rightarrow \quad f'(x) = 2x$$

$$f(x) = x^a \quad \rightarrow \quad f'(x) = ax^{a-1}$$

$$f(x) = cx^a \quad \rightarrow \quad f'(x) = cax^{a-1}$$

$$f(x) = c \quad \rightarrow \quad f'(x) = 0$$

$$f(x) = e^x \quad \rightarrow \quad f'(x) = e^x$$

$$f(x) = \sin(x) \quad \rightarrow \quad f'(x) = \cos(x)$$

$$f(x) = \cos(x) \quad \rightarrow \quad f'(x) = -\sin(x)$$

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- A few simple examples:

$$f(x) = 7x \quad \rightarrow \quad f'(x) = 7$$

$$f(x) = 6x^3 \quad \rightarrow \quad f'(x) = 18x^2$$

$$f(x) = 4x^{-3} \quad \rightarrow \quad f'(x) = -12x^{-4}$$

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- An important property is multiplication of the derivative by a constant:

$$(cf)'(x) = cf'(x)$$

- For example:

$$f(x) = 2x^3 = 2 * (x^3)$$

$$f'(x) = (\,2 * (x^3)\,)' = 2 * (x^3)' = 2 * (3 * x^2) = 6x^2$$

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- The derivative of the sum of two functions:

$$\big(f(x) + g(x)\big)' = f'(x) + g'(x)$$

$$f(x) = \sin(x) \qquad g(x) = x^2$$

$$(\sin(x) + x^2)' = (\sin(x))' + (x^2)' = \cos(x) + 2x$$

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- The derivative of $f(x) = 2x^3 - 3x + 1$:

$$f'(x) \;=\; 2 * 3x^2 - 3 \;=\; 6x^2 - 3$$

- We can also calculate the derivative of a derivative:

$$f''(x) \;=\; \frac{d}{dx} f'(x) \;=\; \frac{d^2}{dx^2} f(x)$$

$$f''(x) \;=\; \frac{d}{dx}(6x^2 - 3) \;=\; (6x^2 - 3)' \;=\; 12x$$

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- The derivative of the product of two functions:

$$\big(f(x) * g(x)\big)' = f'(x)g(x) + f(x)g'(x)$$

$$f(x) = \sin(x) \qquad g(x) = x^2$$

$$(\sin(x)\, x^2)' = (\sin(x))'\, x^2 + \sin(x)\, (x^2)'$$

$$(\sin(x)\, x^2)' = \cos(x)\, x^2 + \sin(x)\, 2x$$

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- All formulas for the sum/difference/product/quotient:

$$\big(f(x)+g(x)\big)' = f(x)' + g(x)'$$

$$\big(f(x)-g(x)\big)' = f(x)' - g(x)'$$

$$\big(f(x)*g(x)\big)' = f'(x)g(x) + f(x)g'(x)$$

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)}$$

Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- The derivative of the composition of two functions:

$$\big(f(g(x))\big)' = f'\big(g(x)\big) * g'(x)$$

- Let's calculate $(\sin(x^2))'$:

$$f(x) = \sin(x) \qquad g(x) = x^2$$

$$\big(f(g(x))\big)' \;=\; (\sin(x^2))' \;=\; \cos(x^2) * 2x$$

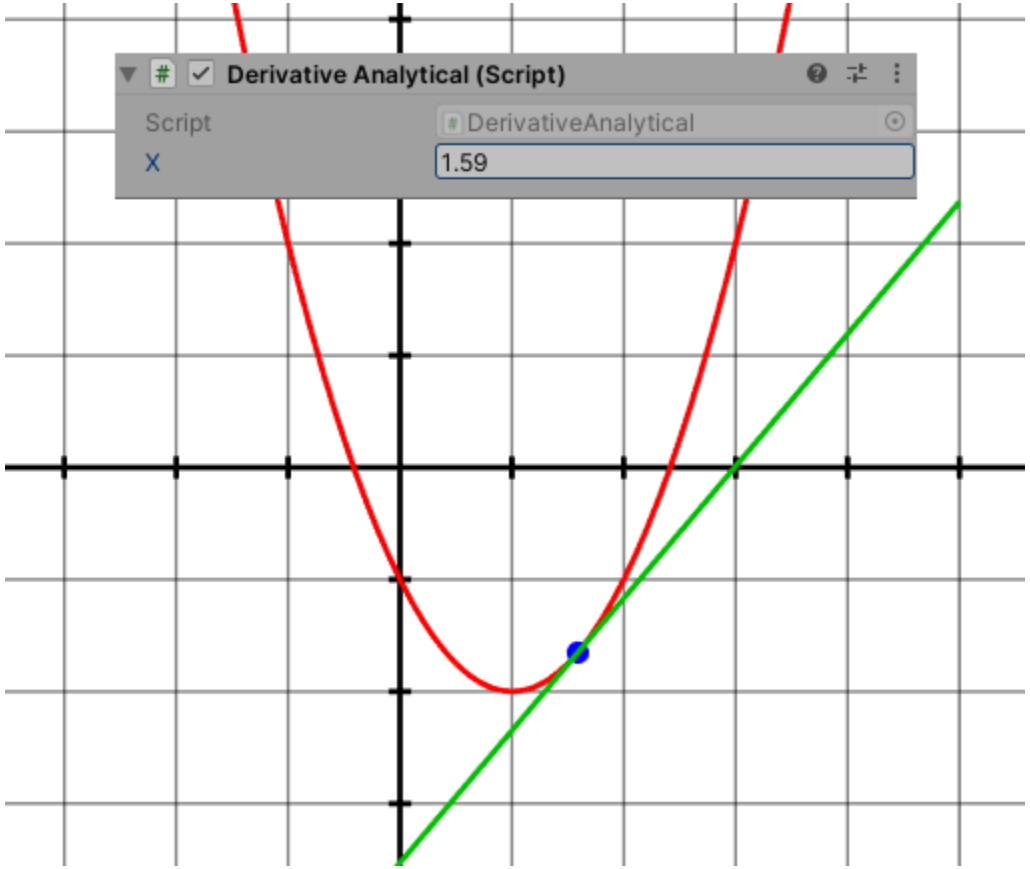
Copyright © by Wojtek Sterna

8.3. Analytical Calculation

- Wolfram

Copyright © by Wojtek Sterna

8.3. Analytical Calculation



Copyright © by Wojtek Sterna

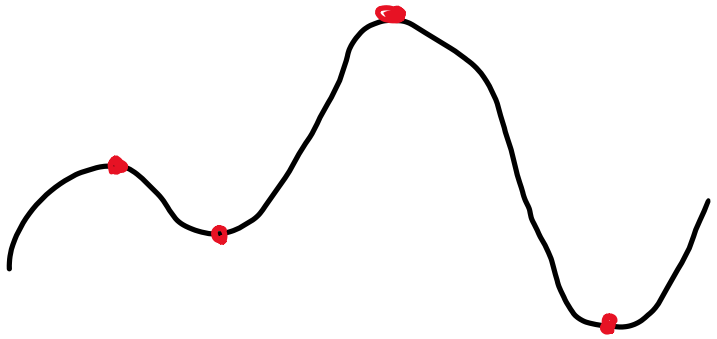
8.3. Analytical Calculation

- The good thing about analytical calculation of the derivative is that it gives us an accurate result
- The problem of analytical calculation is that it is not easy to write code which can calculate the derivative of any function
- A similar situation occurs with calculating the derivative of tabular data. In order to do that we would first need to **approximate** the tabular data with an analytical function and then calculate the derivative of that function

Copyright © by Wojtek Sterna

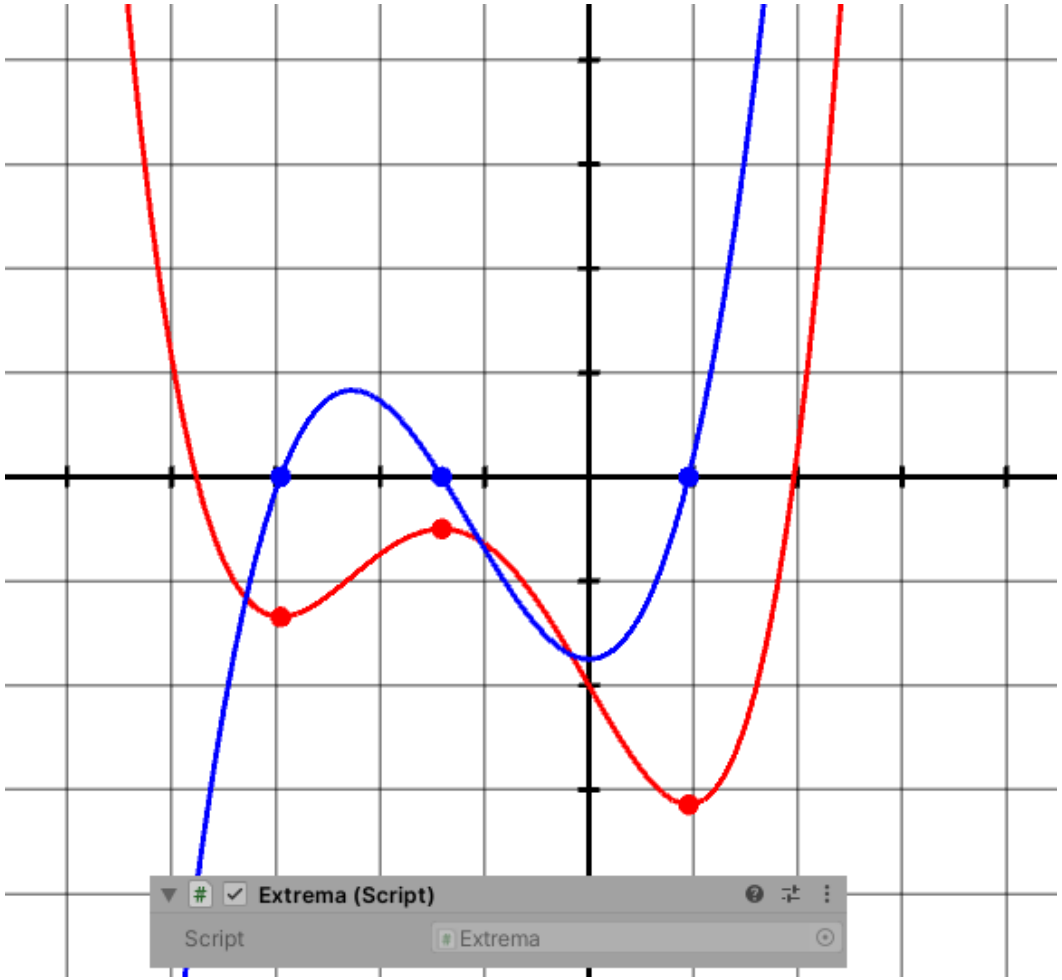
8.4. Function Extrema

- **Extrema** are values where a function takes on the lowest or the highest value
- If the function f takes on the largest value in some neighbourhood we call it a **maximum**. If the lowest – **minimum**
- An extremum can be **local** or **global**
- Best candidates are points whose tangent is parallel to the X axis, i.e. where $f'(x) = 0$. These points are called **critical points**
- Thanks to extrema we can solve a wide array of optimization problems



Copyright © by Wojtek Sterna

8.4. Function Extrema



Copyright © by Wojtek Sterna

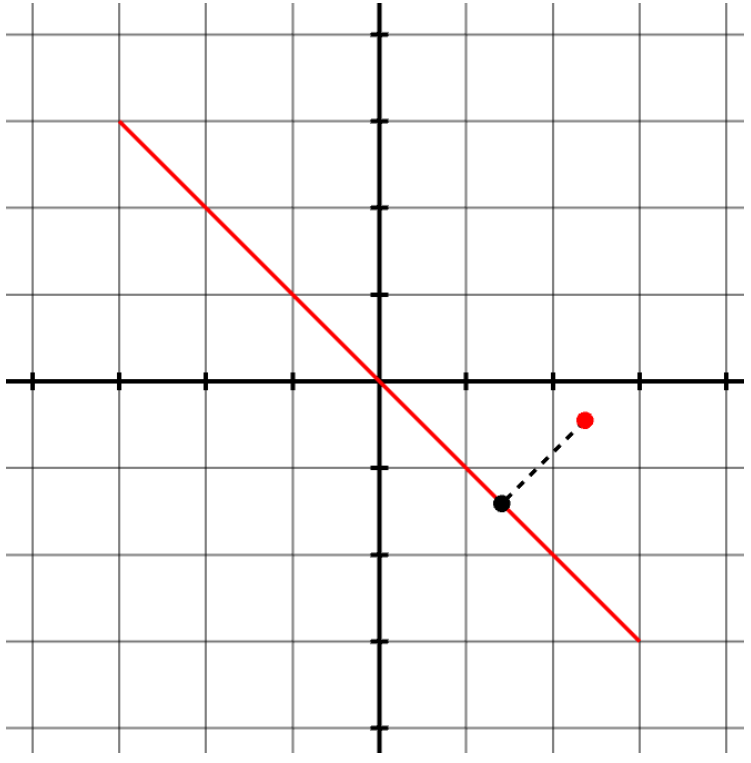
8.4. Function Extrema

- How do we know if an extremum is a minimum or a maximum?
- Oftentimes it is clear from the context of the problem
- A more formal analysis requires examining the neighborhood of a given extremum
- Derivative can also tell us if the function is increasing or decreasing in value
- Beware of **inflection points**!

Copyright © by Wojtek Sterna

8.4. Function Extrema

- We will now write a program that calculates the distance of a point to a parametric line
- The same problem we already solved – using another method (vector projection) – in chapter 4
- In order to calculate the distance of the red point from the line we first find a point on the line that is closest to the red point:



Copyright © by Wojtek Sterna

8.4. Function Extrema

- The parametric line equation:

$$\begin{cases} x(t) = x_0 + \vec{v}_x t \\ y(t) = y_0 + \vec{v}_y t \end{cases}$$

- Point $p = (p_x, p_y)$
- Distance function of the point p to the line:

$$f(t) = \sqrt{\left((x_0 + \vec{v}_x t) - p_x\right)^2 + \left(\left(y_0 + \vec{v}_y t\right) - p_y\right)^2}$$

Copyright © by Wojtek Sterna

8.4. Function Extrema

- There exists a value t, a point on the line, for whom the distance from p is the **lowest**. Function $f(t)$ takes on the smallest value there
- If we calculate:

$$f'(t) = 0$$

we will find out what t that is

Copyright © by Wojtek Sterna

8.4. Function Extrema

- Calculating the derivative will be easier if we get rid off of the square root:

$$f(t) = \left((x_0 + \vec{v}_x t) - p_x\right)^2 + \left(\left(y_0 + \vec{v}_y t\right) - p_y\right)^2$$

  It does not change the nature of the problem because now we deal with the distance squared

- The derivative:

$$f'(t) = 2\vec{v}_x\left((x_0 + \vec{v}_x t) - p_x\right) + 2\vec{v}_y\left(\left(y_0 + \vec{v}_y t\right) - p_y\right)$$

Copyright © by Wojtek Sterna

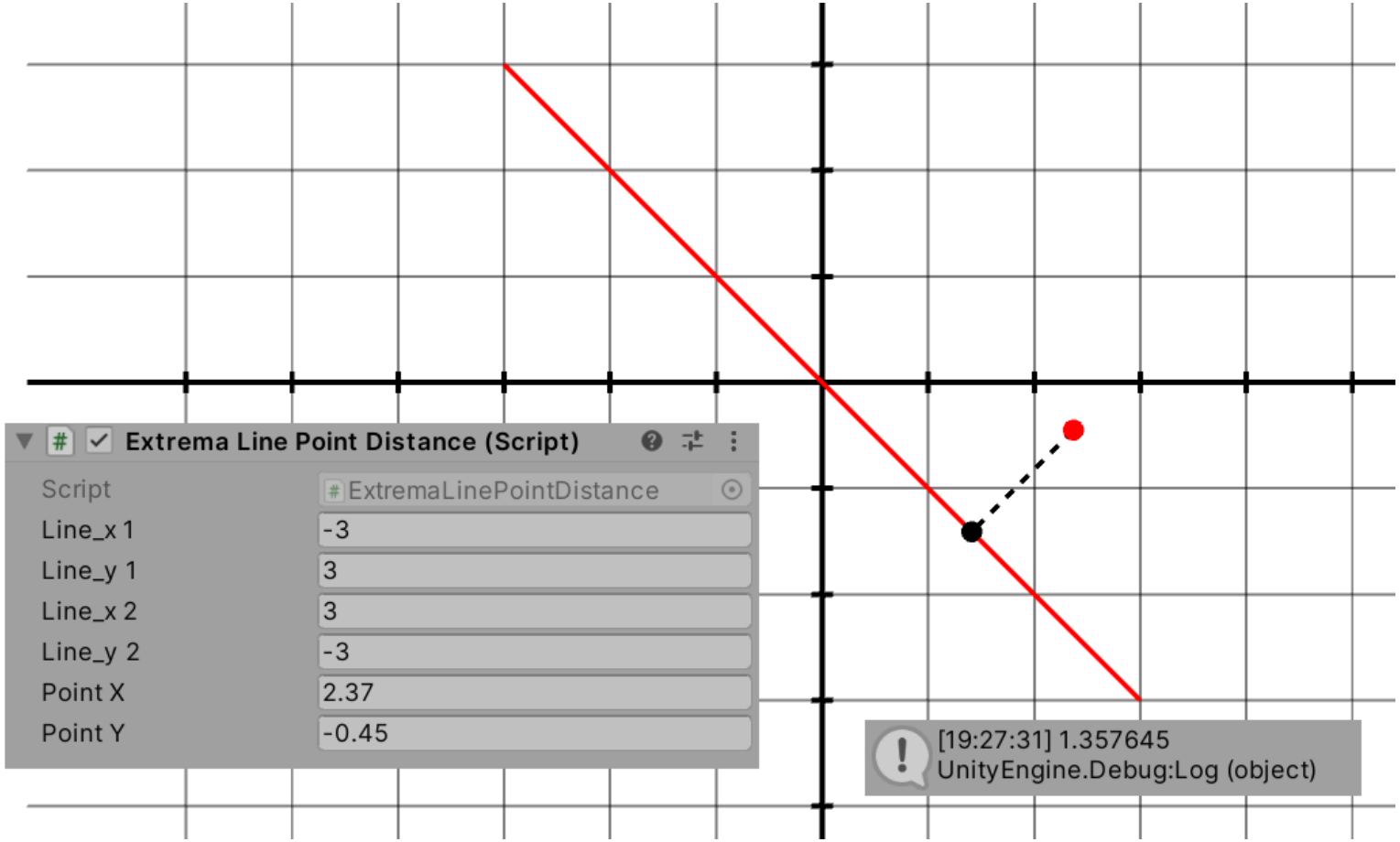
8.4. Function Extrema

- After solving $f'(t) = 0$ we get:

$$t = \frac{p_x \vec{v}_x + p_y \vec{v}_y - \vec{v}_x x_0 - \vec{v}_y y_0}{\vec{v}_x{}^2 + \vec{v}_y{}^2}$$

Copyright © by Wojtek Sterna

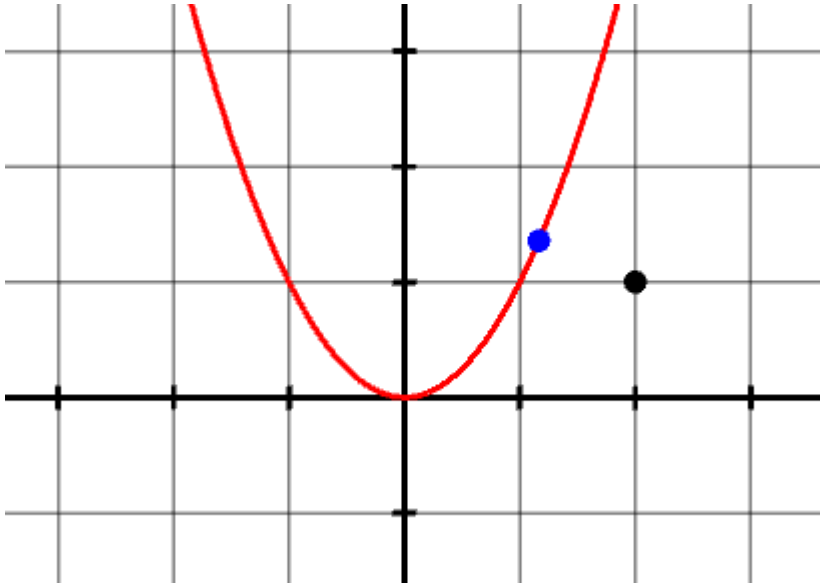
8.4. Function Extrema



Copyright © by Wojtek Sterna

8.4. Function Extrema

- We will solve now a similar problem – given the black point and a parabola we want to find the point on the parabola that is closest to the black point:



Copyright © by Wojtek Sterna

8.4. Function Extrema

- The parabola equation:

$$y = ax^2 + bx + c$$

- Point $p = (p_x, p_y)$
- Squared distance function of the point p to the parabola:

$$d(x) = (x - p_x)^2 + \left(y - p_y\right)^2$$

$$d(x) = (x - p_x)^2 + \left((ax^2 + bx + c\,) - p_y\right)^2$$

Copyright © by Wojtek Sterna

8.4. Function Extrema

- Again, we are calculating the derivative $d'(x)$ and we need to solve $d'(x) = 0$
- The derivative (Wolfram):

$$d'(x) = 2\left((2ax + b)\left(ax^2 + bx + c - p_y\right) + x - p_x\right)$$

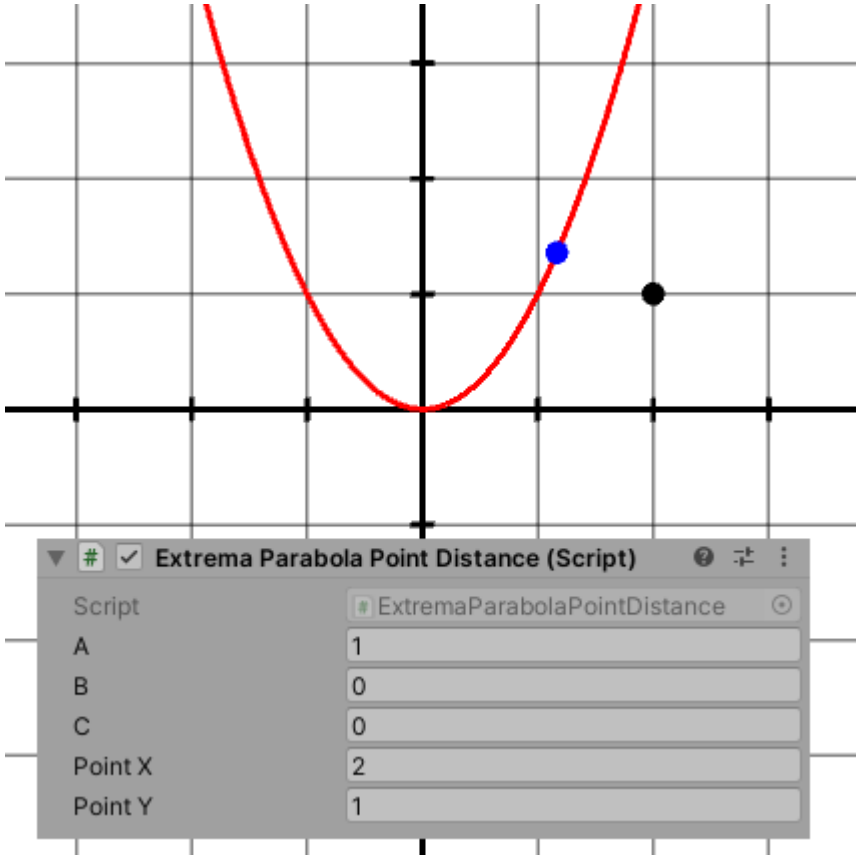
- We are solving (Wolfram):

$$2\left((2a\boldsymbol{x} + b)\left(a\boldsymbol{x}^2 + b\boldsymbol{x} + c - p_y\right) + \boldsymbol{x} - p_x\right) = 0$$

- We will only take the first result (whose solution is in real numbers) out of the three (the remaining two have solutions in complex numbers)

Copyright © by Wojtek Sterna

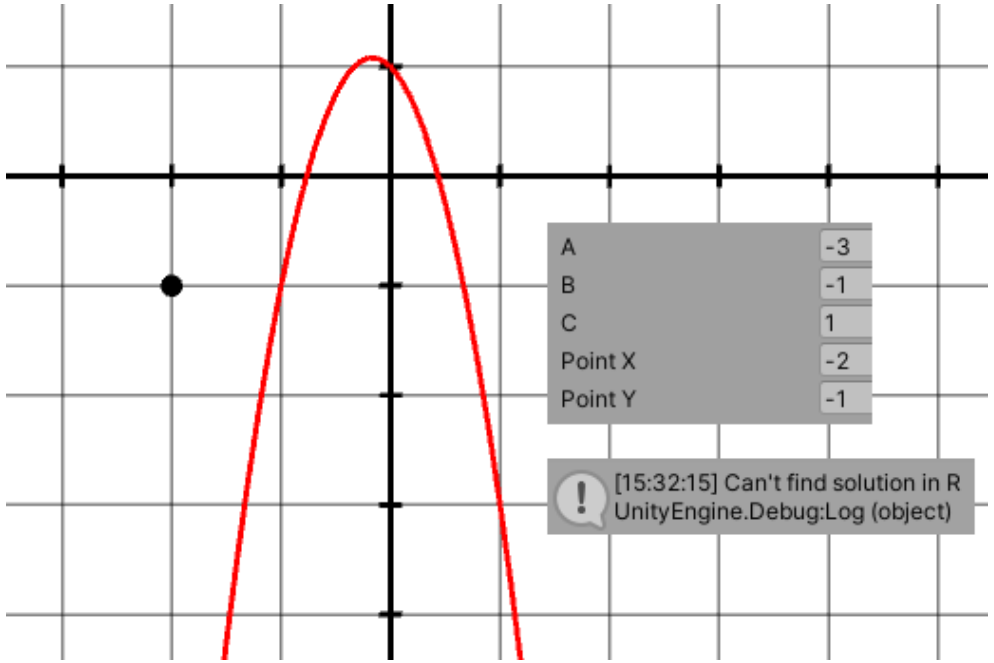
8.4. Function Extrema



Copyright © by Wojtek Sterna

8.4. Function Extrema

- By experimenting with the program we will notice that not every set of parameters gives us a solution:



- Obviously, a solution exists in this case: [Wolfram]

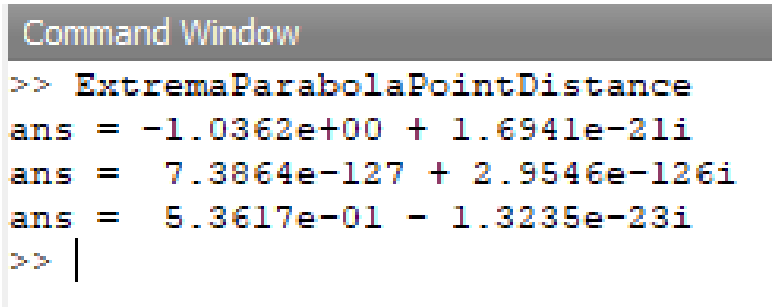
Copyright © by Wojtek Sterna

8.4. Function Extrema

- It turns out that:
  1) we should not have discarded the other two solutions
  2) we should have calculated all solutions in the domain of complex numbers
- Despite the fact that the results are complex numbers, after substituting actual values we will end up with real numbers (thanks to utilizing properties like $i^2 = -1$)
- Since we do not have built-in support for complex numbers in C#/Unity, we will not improve this program, although we will come back to it later

Copyright © by Wojtek Sterna

8.4. Function Extrema



Copyright © by Wojtek Sterna

8.4. Function Extrema

- Note that in the problem being discussed we start in the domain of real numbers and eventually we end up with real numbers, DESPITE the fact that along the way (to the solution), in calculations, we deal with complex numbers
- We deal with similar situations every day. For example in the following equation:

$$x^2 - 2x - 3 = 0$$

only **integer numbers** occur. Its solution are also **integer numbers**:

$$x = -1 \quad \text{or} \quad x = 3$$

However, to get there we had to use **rational numbers**:

$$x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

Copyright © by Wojtek Sterna

8.5. Derivative of Parametric Function

- We know from chapter 4 that geometrical objects can be represented with parametric equations
- Parametric equation usually consists of more than one equation. For example:

$$\begin{cases} x(t) = x_0 + \vec{v}_x t \\ y(t) = y_0 + \vec{v}_y t \end{cases}$$

  is the parametric equation of 2D line
- If we calculate the derivatives of all these equations we will get the **tangent vector** to the object that is represented with these equations:

$$\vec{t} = [x'(t), y'(t)]$$

Copyright © by Wojtek Sterna

8.5. Derivative of Parametric Function

- The parametric 2D line:

$$\begin{cases} x(t) = x_0 + \vec{v}_x t \\ y(t) = y_0 + \vec{v}_y t \end{cases}$$

- Its derivatives:

$$\begin{cases} x'(t) = \vec{v}_x \\ y'(t) = \vec{v}_y \end{cases}$$

- The tangent vector:

$$\vec{t} \;=\; [x'(t), y'(t)] \;=\; \left[\vec{v}_x, \vec{v}_y\right]$$

Copyright © by Wojtek Sterna

8.5. Derivative of Parametric Function

- The parametric circle:

$$\begin{cases} x(\theta) = a + r\cos(\theta) \\ y(\theta) = b + r\sin(\theta) \end{cases}$$

- Its derivatives:

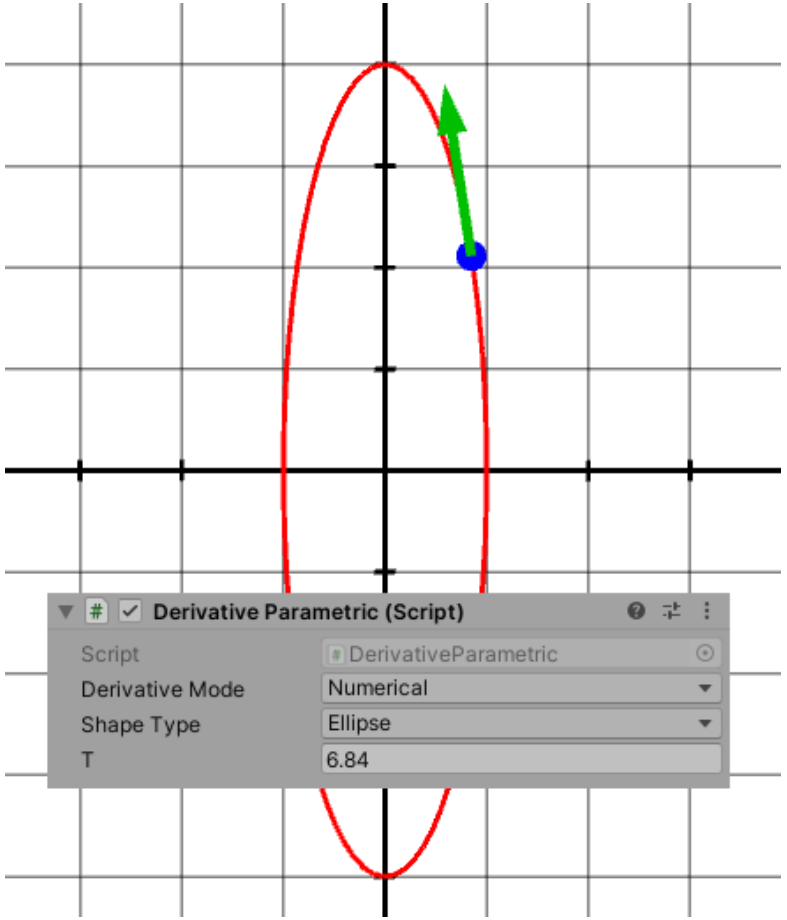
$$\begin{cases} x'(\theta) = -r\sin(\theta) \\ y'(\theta) = r\cos(\theta) \end{cases}$$

- The tangent vector:

$$\vec{t} = [x'(\theta), y'(\theta)] = [-r\sin(\theta), r\cos(\theta)]$$

Copyright © by Wojtek Sterna

8.5. Derivative of Parametric Function



Copyright © by Wojtek Sterna

8.5. Derivative of Parametric Function

- Note that in 2D it is easy to find the normal vector to such parametric function
- If we have the tangent vector:

$$\vec{t} = [x'(\theta), y'(\theta)]$$

then the normal vector is:

$$\vec{n}_1 = [-y'(\theta), x'(\theta)] \qquad \text{or} \qquad \vec{n}_2 = [y'(\theta), -x'(\theta)]$$

Copyright © by Wojtek Sterna

8.5. Derivative of Parametric Function

- Consider a special case, where $x(t) = t$:

$$\begin{cases} x(t) = t \\ y(t) = \cos(t) \end{cases}$$

- Then:

$$\begin{cases} x'(t) = 1 \\ y'(t) = -\sin(t) \end{cases}$$

- So the tangent vector is:

$$\vec{t} = [1, -\sin(t)]$$

Copyright © by Wojtek Sterna

8.5. Derivative of Parametric Function

- In the case where $x(t) = t$, that is $\boldsymbol{x} = \boldsymbol{t}$, the parametric equation is reduced to a simple function of one variable:

$$y(\boldsymbol{t}) \;=\; y(\boldsymbol{x}) \;=\; \cos(\boldsymbol{x})$$

- As such, always when we have a function of one variable its tangent vector at x is:

$$\vec{t}(x) = [1, y'(x)]$$

Copyright © by Wojtek Sterna

8.5. Derivative of Parametric Function

- To sum up: when thinking about the derivative we should immediately have two classes of problems in mind, which we can tackle using derivatives:

  1) Calculating tangent vectors

  2) Solving certain optimization problems

Copyright © by Wojtek Sterna

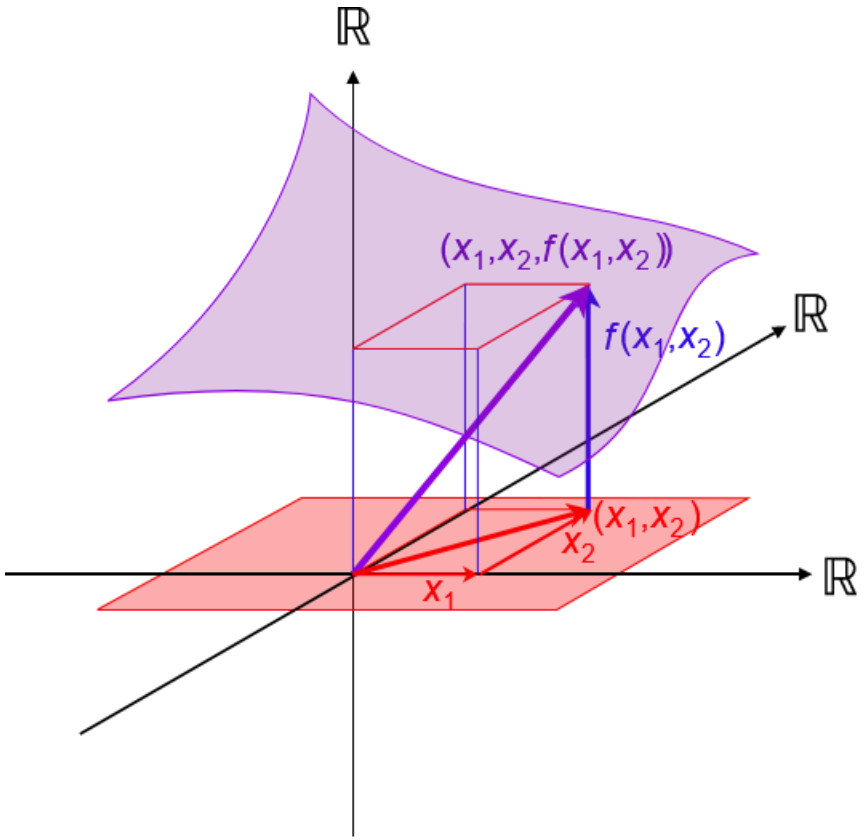
8.6. Partial Derivatives and Gradient

- The graph of a function of two variables (e.g. x and y) is a **surface**:

$$z = f(x, y)$$

- Each point on a surface has three coordinates:

$$(x, y, z) = (x, y, f(x, y))$$



https://en.wikipedia.org/wiki/File:Real_function_of_two_real_variables.svg **Author** Maschen

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- When dealing with a function of more than one variable we can calculate more than one derivative
- For example, let's consider the function:

$$f(x, y) = x^2 + 3xy$$

- Here we can calculate the derivative for both x and y. When we calculate the derivative of a selected variable, the other variables are treated as constants
- The derivative with respect to one variable is called **partial derivative** and is denoted as:

$$f_x$$

$$\frac{\partial f}{\partial x} \quad \text{or} \quad \frac{\partial}{\partial x} f$$

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- Let's calculate all derivatives of the function we've just seen:

$$f(x, y) = x^2 + 3xy$$

$$f_x = 2x + 3y$$

$$f_y = 3x$$

- We can keep on differentiating; for example the second derivative with respect to x:

$$f_{xx} = \frac{\partial^2 f}{\partial x^2} = 2$$

Copyright © by Wojtek Sterna

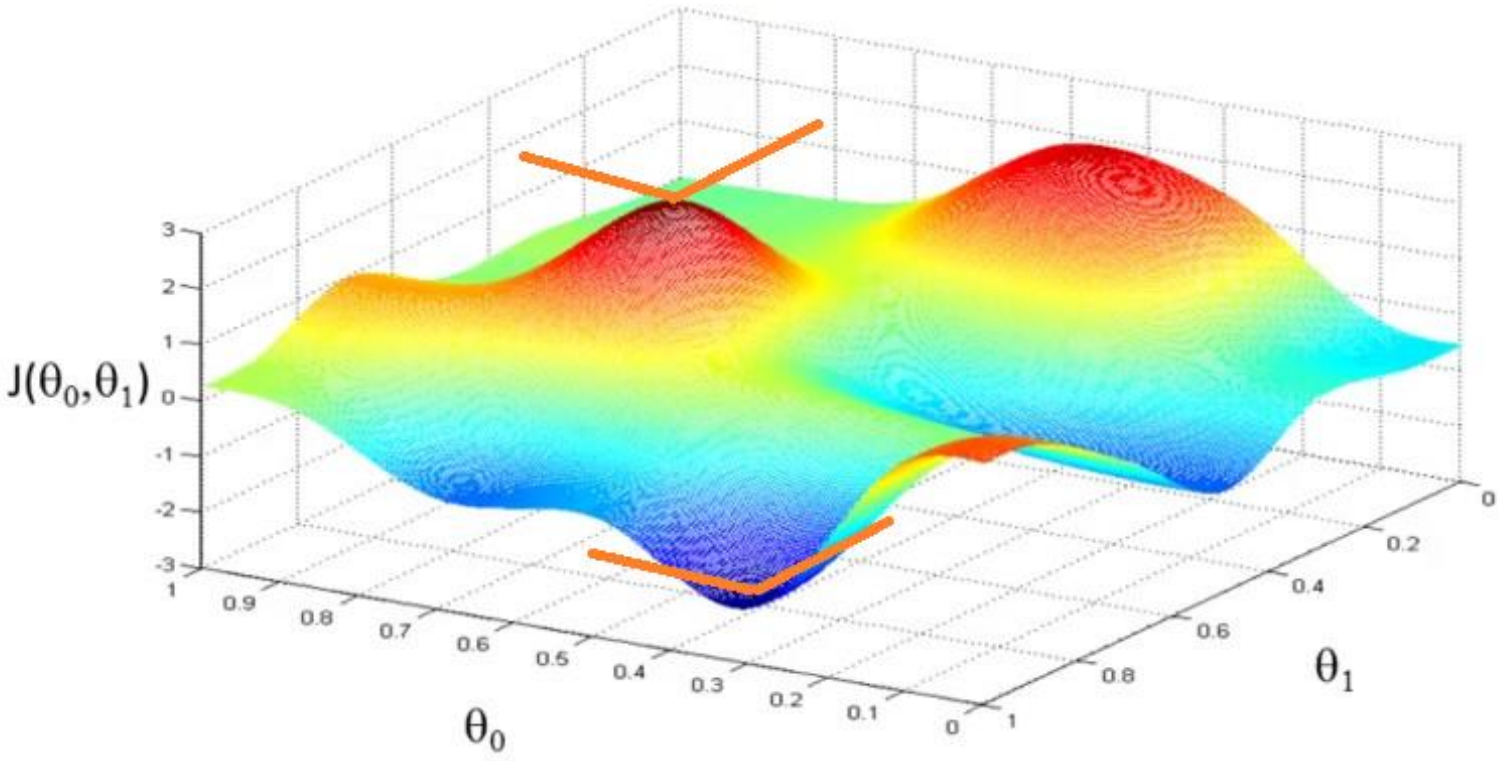
8.6. Partial Derivatives and Gradient

- By finding the x value for which the derivative was 0 we were able to find an extremum of a function
- It is similar when we have a function of several variables. When all partial derivatives are 0 then we might have found an extremum

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- An example minimum and maximum:

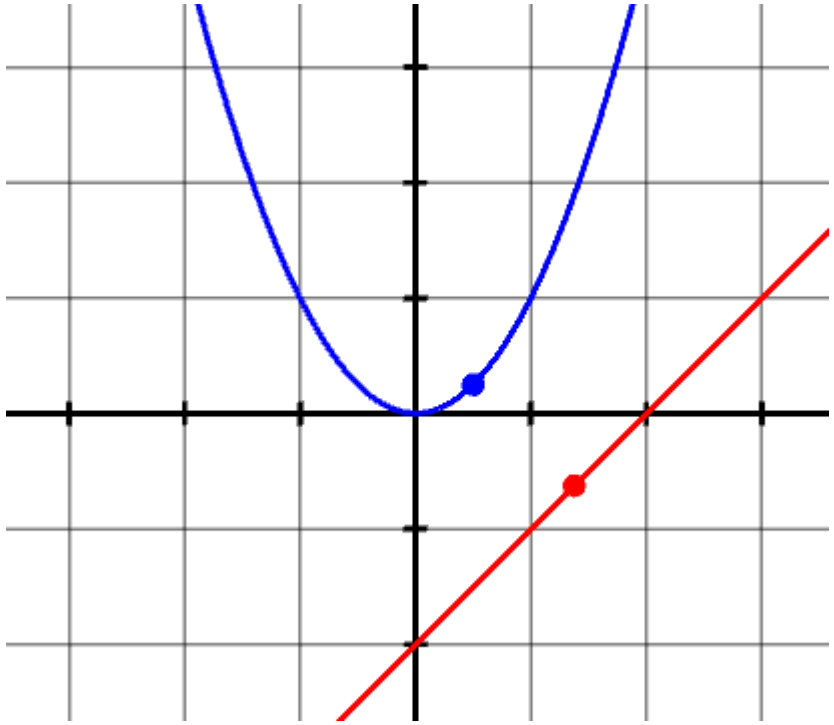


https://www.internalpointers.com/post/gradient-descent-function

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- Let’s consider the following problem: given the equations of a line and a parabola, we want to find the points on the line and the parabola between which the distance is the smallest:



Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- We start the same way as always, which is describing the objects with equations. The line (here we chose the linear equation but we could also use the other forms):

$$y = a_1 x + b_1$$

  and the parabola:

$$y = a_2 x^2 + b_2 x + c_2$$

- Somewhere on the line there is a point $p_1 = (x_1, y_1)$, while on the parabola there is a point $p_2 = (x_2, y_2)$, which are the solution to our problem

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- Our goal is to find such p_1 and p_2, for whom the distance between those points will be the smallest
- We need to start with a function that returns the distance between those two points:

$$d(x_1, x_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$y_1 = a_1 x_1 + b_1$$

$$y_2 = a_2 {x_2}^2 + b_2 x_2 + c_2$$

- It will be easier to work with the squared distance function (as before):

$$d(x_1, x_2) = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- Let's substitute y_1 and y_2:

$$d(x_1, x_2) = (x_1 - x_2)^2 + \left((a_1 x_1 + b_1) - (a_2 {x_2}^2 + b_2 x_2 + c_2)\right)^2$$

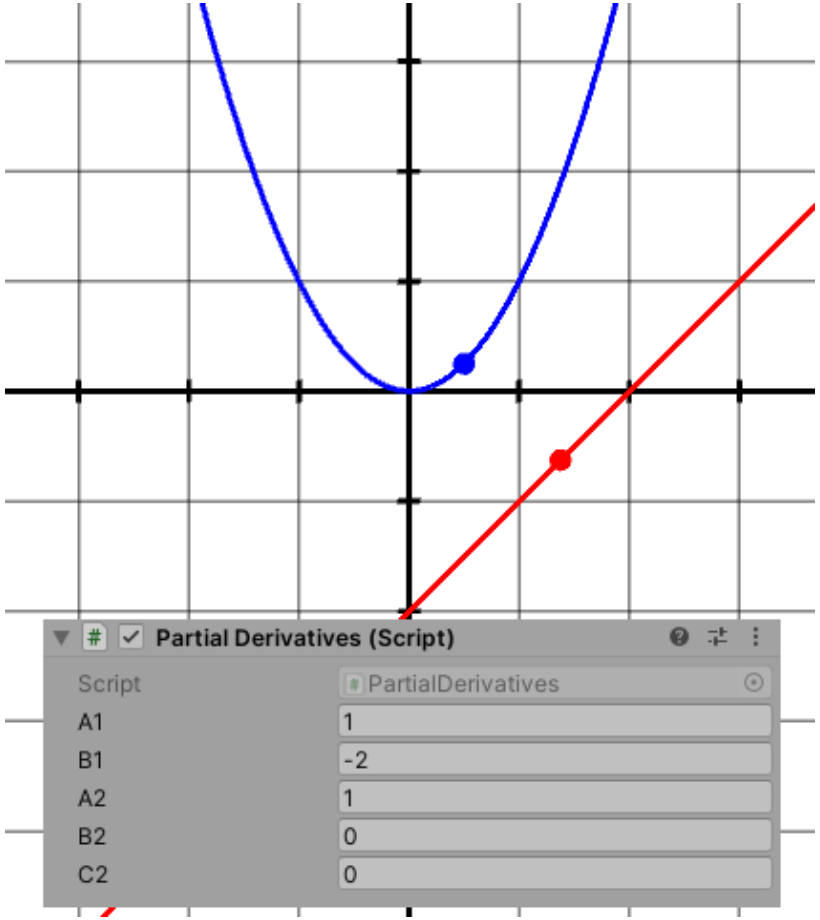
- We have a function of two variables d. If we calculate its derivatives d_{x_1} and d_{x_2}, and then equate them to 0 and solve, we will have found the minimum of d:

$$\begin{cases} d_{x_1} = 2\left(\boldsymbol{x_1} - \boldsymbol{x_2} + a_1(b_1 - c_2 + a_1\boldsymbol{x_1} - b_2\boldsymbol{x_2} - a_2{\boldsymbol{x_2}}^2)\right) = 0 \\ d_{x_2} = 2\left(-\boldsymbol{x_1} + \boldsymbol{x_2} - (b_2 + 2a_2\boldsymbol{x_2})(b_1 - c_2 + a_1\boldsymbol{x_1} - b_2\boldsymbol{x_2} - a_2{\boldsymbol{x_2}}^2)\right) = 0 \end{cases}$$

- Wolfram (d_{x_1}), Wolfram (d_{x_2}) and Wolfram (the system of equations)

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient



Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- If we take all the partial derivatives of a function and make up a vector out of them we will get the so-called **gradient**:

$$f_x = 2x + 3y$$

$$f_y = 3x$$

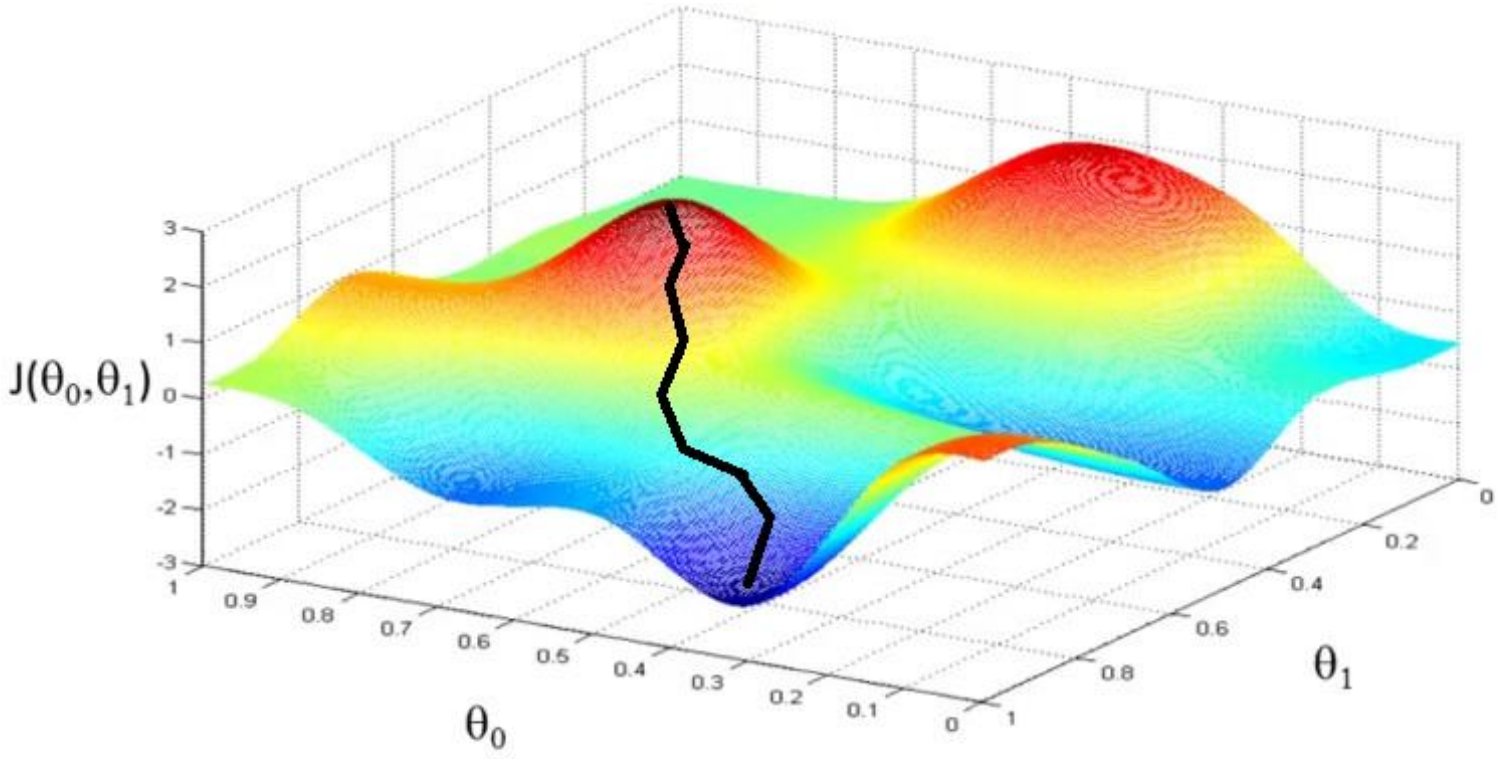
$$\nabla f(x, y) = [f_x, f_y] = [2x + 3y, 3x]$$

- Gradient can be calculated at any point and it determines the direction along which the function's value increases the most. By going along that direction we have a good chance of reaching a maximum

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- An example path to a maximum:



https://www.internalpointers.com/post/gradient-descent-function

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient

- Each point on a surface has three coordinates:

$$(x, y, z) = (x, y, f(x, y))$$

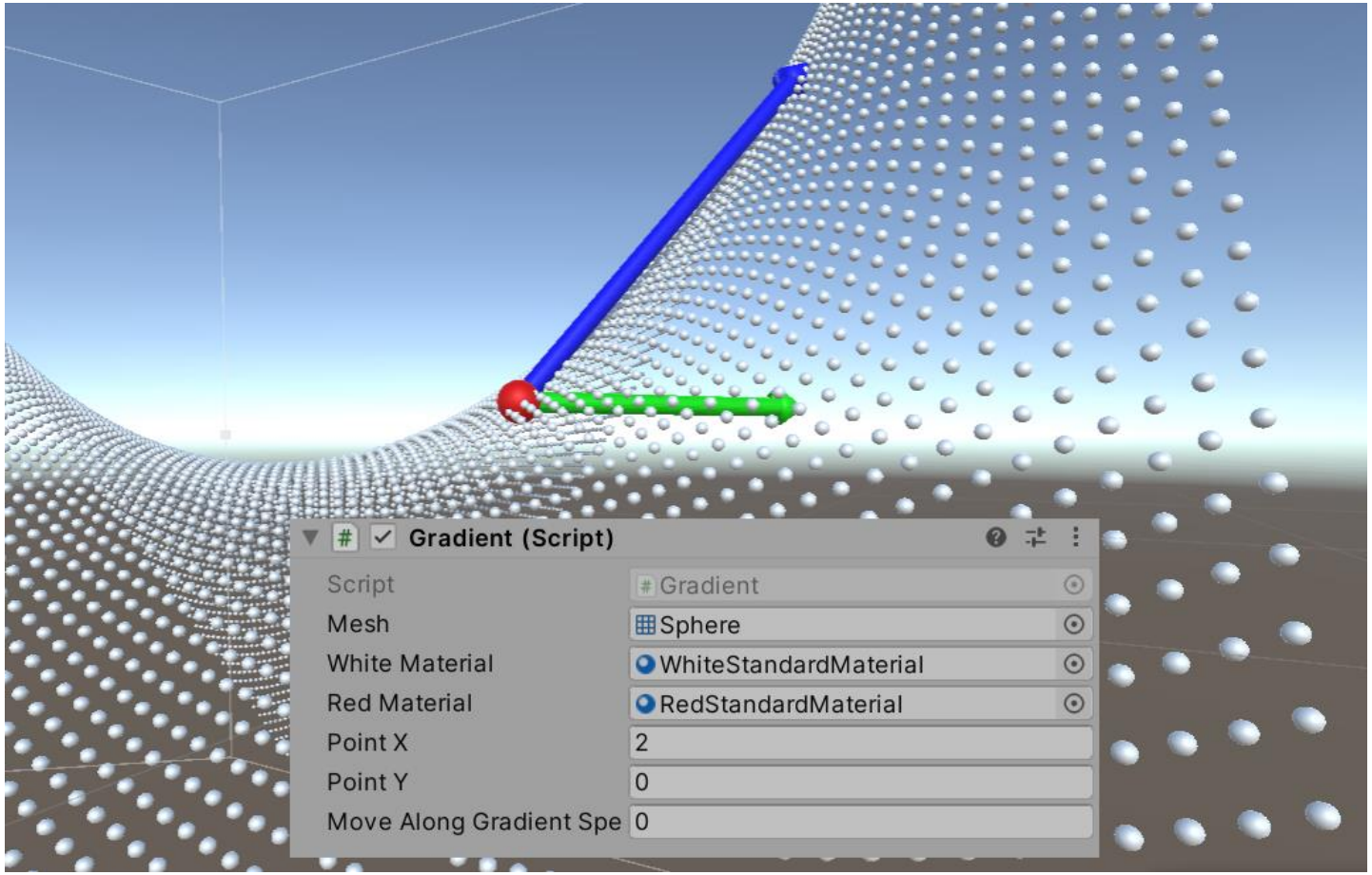
- However, the gradient in each point is **two-dimensional**:

$$\nabla f(x, y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- It stems from the fact that the gradient tells us by how much to change the arguments that go into the function.
  In this case we have two arguments

Copyright © by Wojtek Sterna

8.6. Partial Derivatives and Gradient



Copyright © by Wojtek Sterna

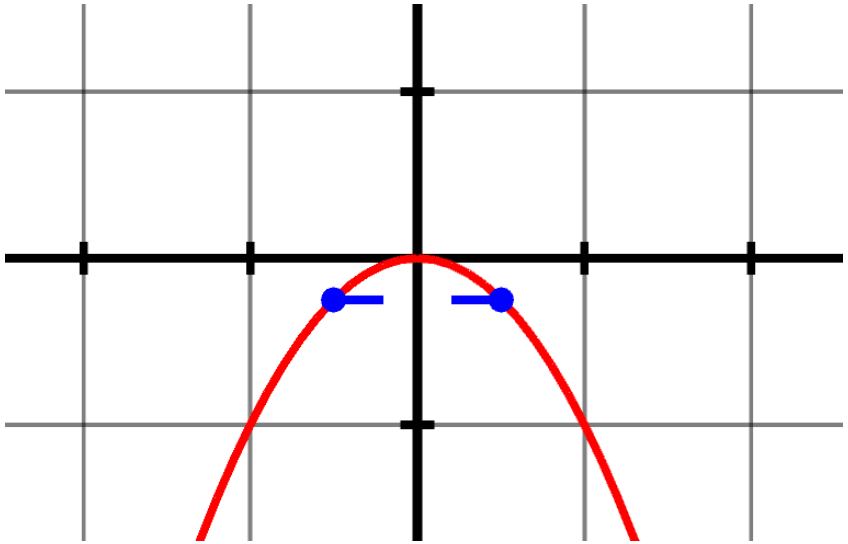
8.6. Partial Derivatives and Gradient

- For a 3D surface the gradient is 2D
- For a 2D graph the gradient is 1D:

$$(x, y) = \big(x, f(x)\big)$$

$$\nabla f(x) = \left[\frac{\partial f}{\partial x}\right] = [f'(x)]$$

$$f(x) = -x^2 \qquad f'(x) = -2x$$



Copyright © by Wojtek Sterna

8.7. Derivative of Implicit Function

- We know from chapter 4 that geometrical objects can be represented with implicit equations
- The implicit circle equation:

$$(x-a)^2+(y-b)^2-r^2=0$$

- This is actually a function of two variables:

$$f(x,y)=(x-a)^2+(y-b)^2-r^2$$

Copyright © by Wojtek Sterna

8.7. Derivative of Implicit Function

- We can calculate the partial derivatives of that function:

$$f(x,y) = (x-a)^2 + (y-b)^2 - r^2$$

$$f(x,y) = x^2 + a^2 - 2ax + y^2 + b^2 - 2by - r^2$$

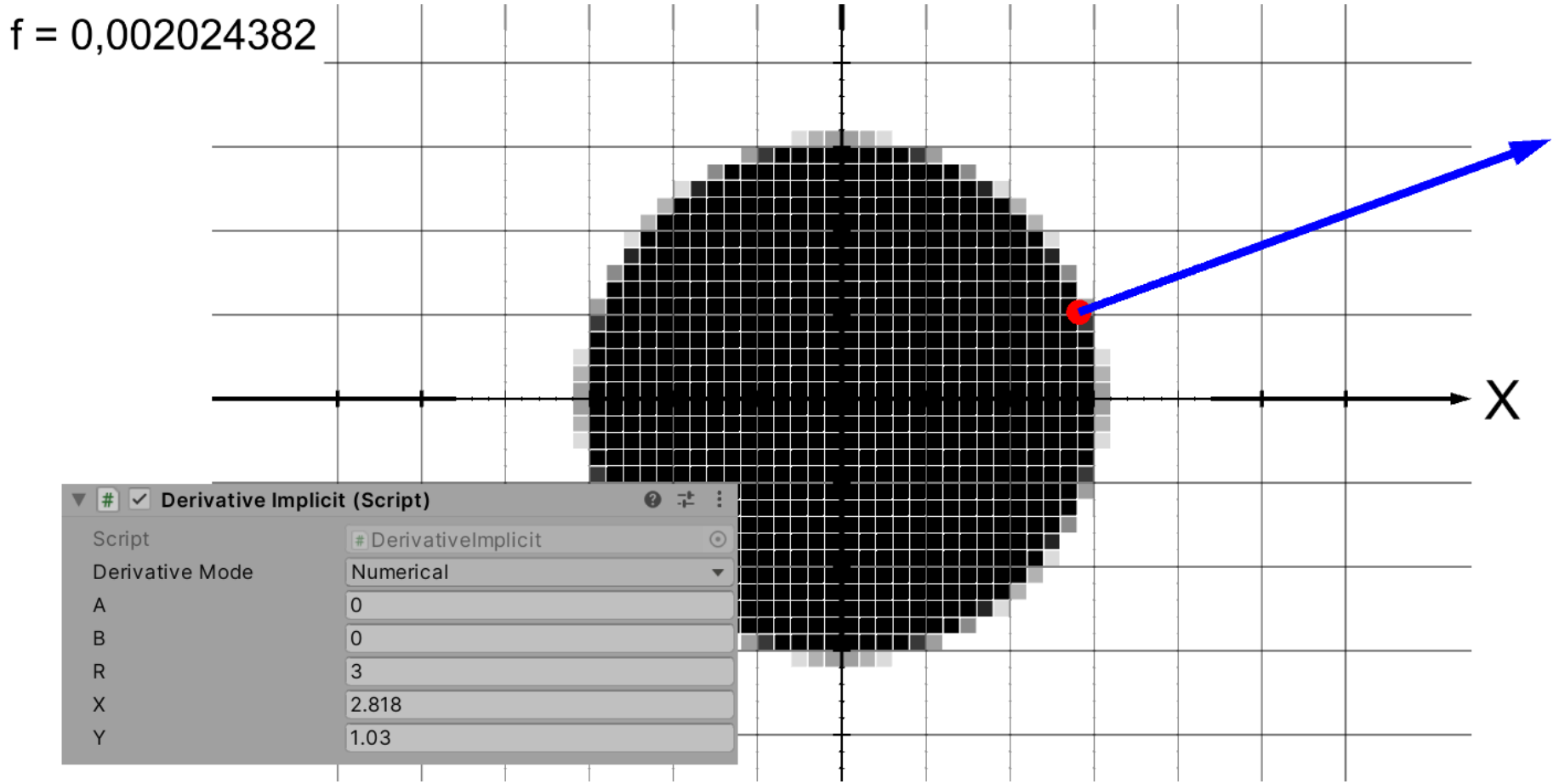
$$f_x = 2x - 2a$$

$$f_y = 2y - 2b$$

- In case of an implicit function, when we take all its partial derivatives (gradient) we will get the normal vector to the object represented with that implicit function

Copyright © by Wojtek Sterna

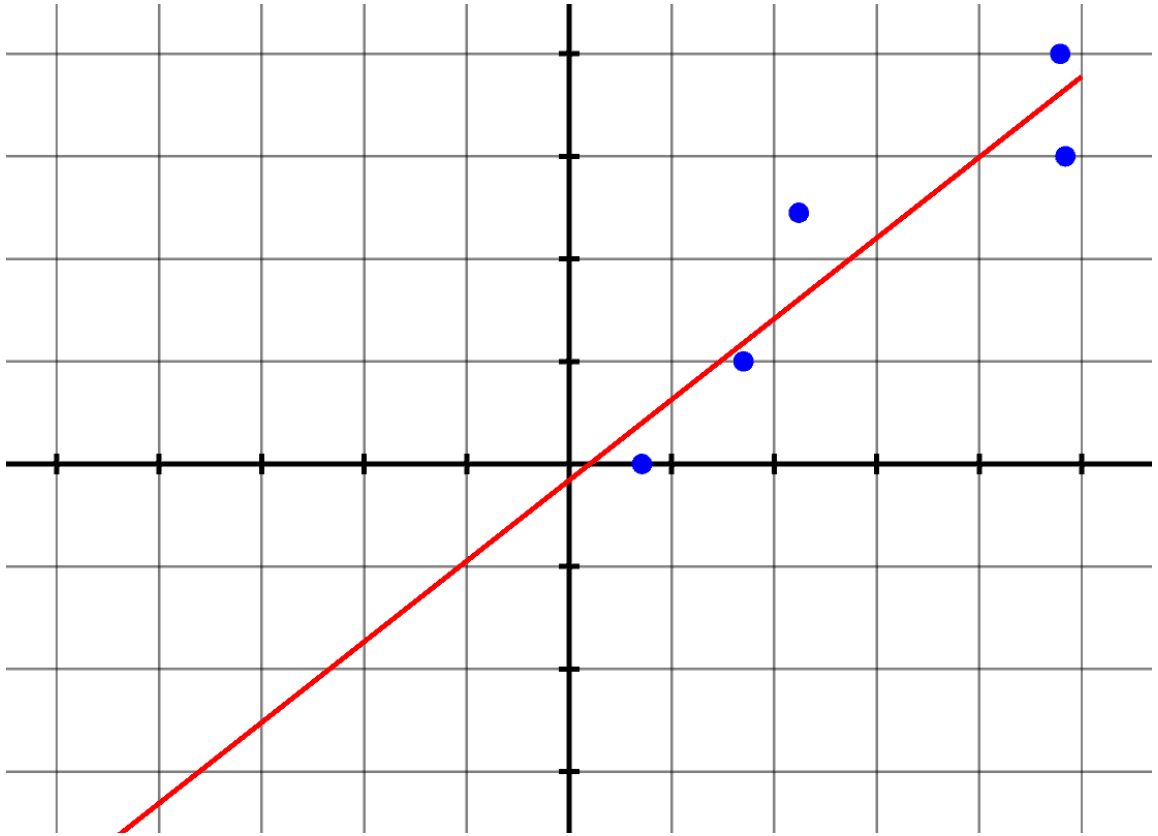
8.7. Derivative of Implicit Function



Copyright © by Wojtek Sterna

8.8. Least Squares Method

- Let’s say we want to solve the following optimization problem. We want to find the equation of a straight line that best approximates a certain set of points:



Copyright © by Wojtek Sterna

8.8. Least Squares Method

- That particular problem is called **linear regression**
- One way to solve this problem is via the **least squares method**

Copyright © by Wojtek Sterna

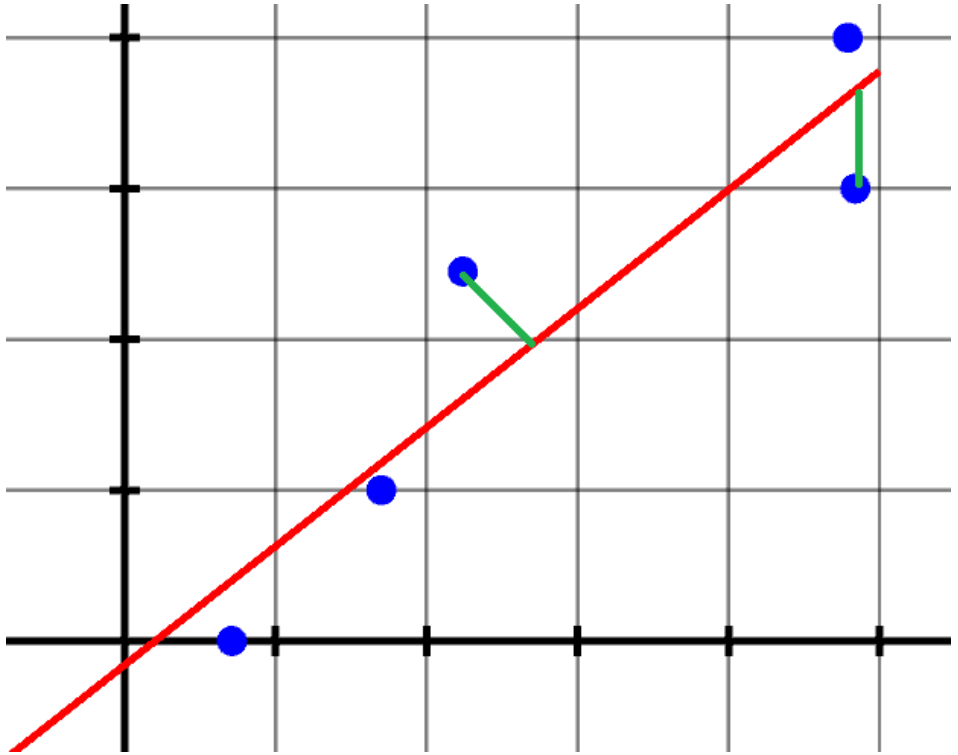
8.8. Least Squares Method

- Let’s start with defining the optimization problem
- We want the **sum of distances** of points from the line to be as small as possible
- Our goal is to find the straight line equation, that is the coefficients a and b of the linear function $y = ax + b$

Copyright © by Wojtek Sterna

8.8. Least Squares Method

- The distance of a point from a line can be calculated in two ways:



Copyright © by Wojtek Sterna

8.8. Least Squares Method

- The line is of form $y = ax + b$
- Let's take some point with coordinates (x_1, y_1)
- Its distance d_1 from the line can be calculated like so:

$$d_1 = |(ax_1 + b) - y_1|$$

- Let's now sum n distances of points:

$$d = \sum_{i=1}^{n} |(ax_i + b) - y_i|$$

Copyright © by Wojtek Sterna

8.8. Least Squares Method

- The formula we have just come up with is actually a function of two variables a and b:

$$d(a,b) = \sum_{i=1}^{n} |(ax_i + b) - y_i|$$

- By arbitrarily picking values of a and b (meaning we get different straight lines) we will be getting different values of the sum of points' distances
- For a certain pair of a and b the value of d will be the smallest of all...

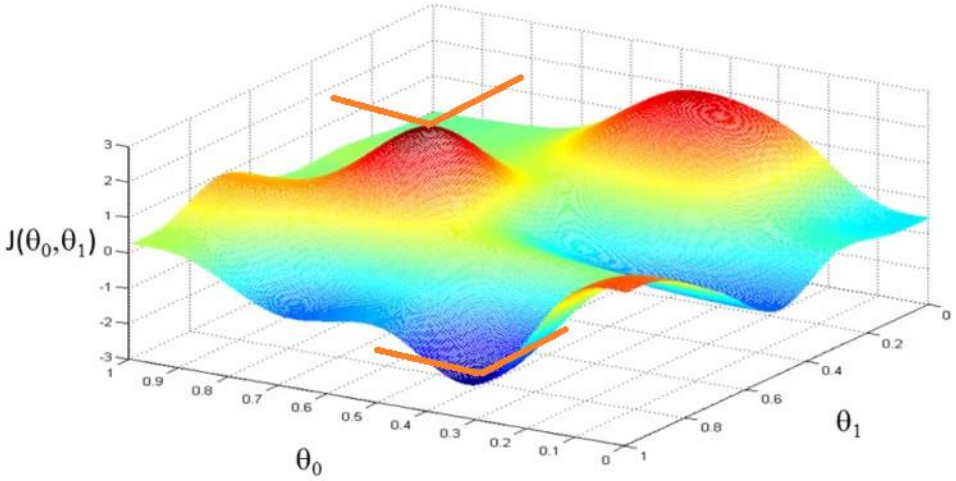
Copyright © by Wojtek Sterna

8.8. Least Squares Method

- We can find the smallest value of d if we find such a and b, for which the partial derivatives of $d(a, b)$ are all 0:

$$d_a = 0$$

$$d_b = 0$$



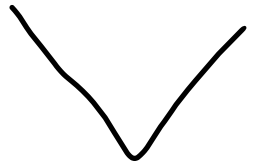
https://www.internalpointers.com/post/gradient-descent-function

Copyright © by Wojtek Sterna

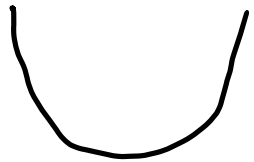
8.8. Least Squares Method

- There is a small problem with our function $d(a, b)$. It uses the absolute value, what makes differentiation problematic
- To avoid this problem we can use the square function instead of the absolute:

$$d(a,b) = \sum_{i=1}^{n} |(ax_i + b) - y_i|$$



$$D(a,b) = \sum_{i=1}^{n} ((ax_i + b) - y_i)^2$$



- That is where the name of the least **squares** method comes from
- Function D is also often called the **cost function**

Copyright © by Wojtek Sterna

8.8. Least Squares Method

- Let’s calculate the partial derivatives:

$$\frac{\partial D}{\partial a} = D_a = \sum_{i=1}^{n} 2 * (ax_i + b - y_i) * (x_i)$$

$$\frac{\partial D}{\partial b} = D_b = \sum_{i=1}^{n} 2 * (ax_i + b - y_i) * (1)$$

- Wolfram
- We are looking for such a and b, where $D_a = 0$ and $D_b = 0$, which is where the „cost” (the value) of D is the smallest

Copyright © by Wojtek Sterna

8.8. Least Squares Method

- This boils down to solving the following system of two equations with two unknowns:

$$\sum_{i=1}^{n} 2 * (\boldsymbol{a}x_i + \boldsymbol{b} - y_i) * (x_i) = 0$$

$$\sum_{i=1}^{n} 2 * (\boldsymbol{a}x_i + \boldsymbol{b} - y_i) * (1) = 0$$

Copyright © by Wojtek Sterna

8.8. Least Squares Method

- After some rearrangements we get:

$$\boldsymbol{a}\sum_{i=1}^{n}(x_i)^2 + \boldsymbol{b}\sum_{i=1}^{n}x_i - \sum_{i=1}^{n}x_i y_i = 0$$

$$\boldsymbol{a}\sum_{i=1}^{n}x_i + \boldsymbol{b}n - \sum_{i=1}^{n}y_i = 0$$

Copyright © by Wojtek Sterna

8.8. Least Squares Method

- Let's solve:

$$\boldsymbol{a}A + \boldsymbol{b}B + C = 0$$
$$\boldsymbol{a}D + \boldsymbol{b}E + F = 0$$

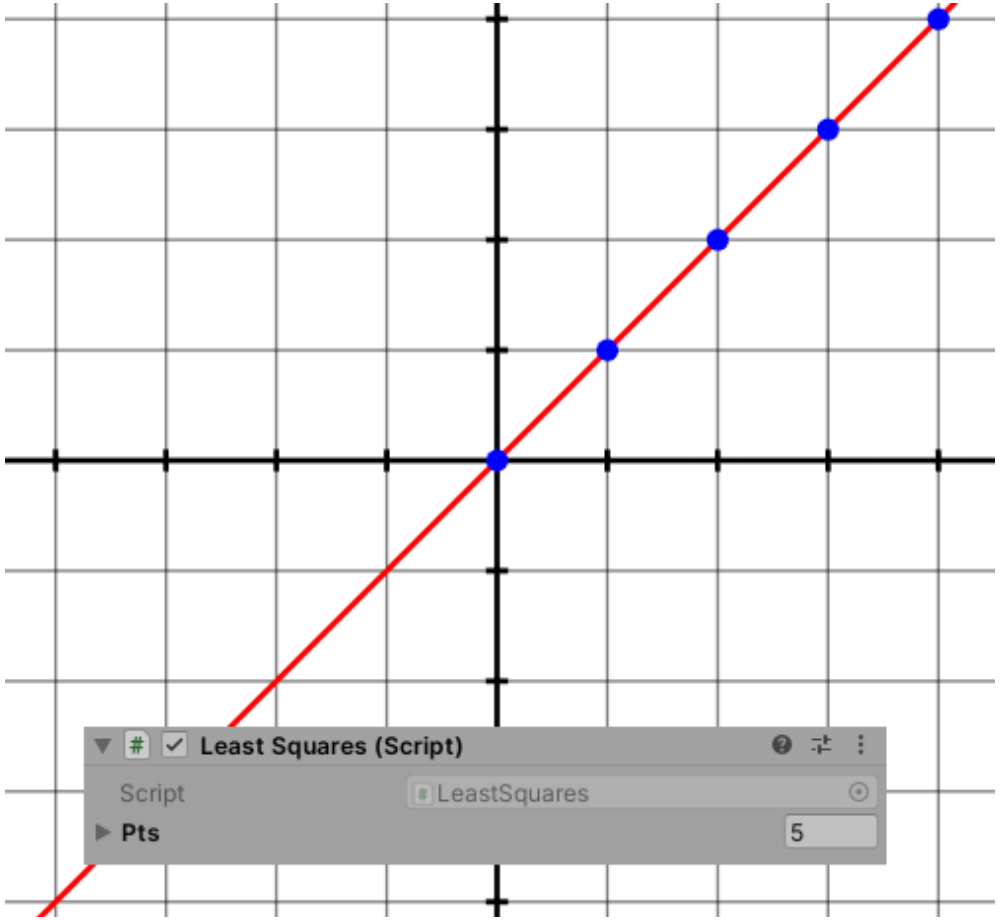
$$A = \sum_{i=1}^{n} (x_i)^2 \qquad B = \sum_{i=1}^{n} x_i \qquad C = -\sum_{i=1}^{n} x_i y_i$$

$$D = \sum_{i=1}^{n} x_i \qquad E = n \qquad F = -\sum_{i=1}^{n} y_i$$

- Wolfram

Copyright © by Wojtek Sterna

8.8. Least Squares Method



Copyright © by Wojtek Sterna

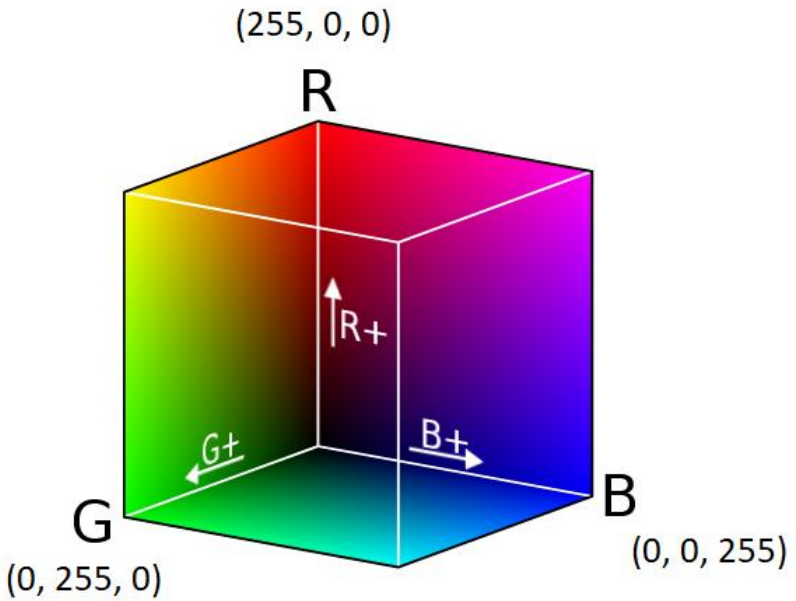
8.8. Least Squares Method

- Overall finding arguments of a function for which the function's derivatives are 0 means we've likely found a minimum or a maximum
- The cost function is a function whose smallest possible value is 0. Ideally this function is continuous and differentiable – for example thanks to squaring the absolute value
- The idea of the least squares method is to find such function's arguments for which the function reaches the global minimum („at best" it is 0)

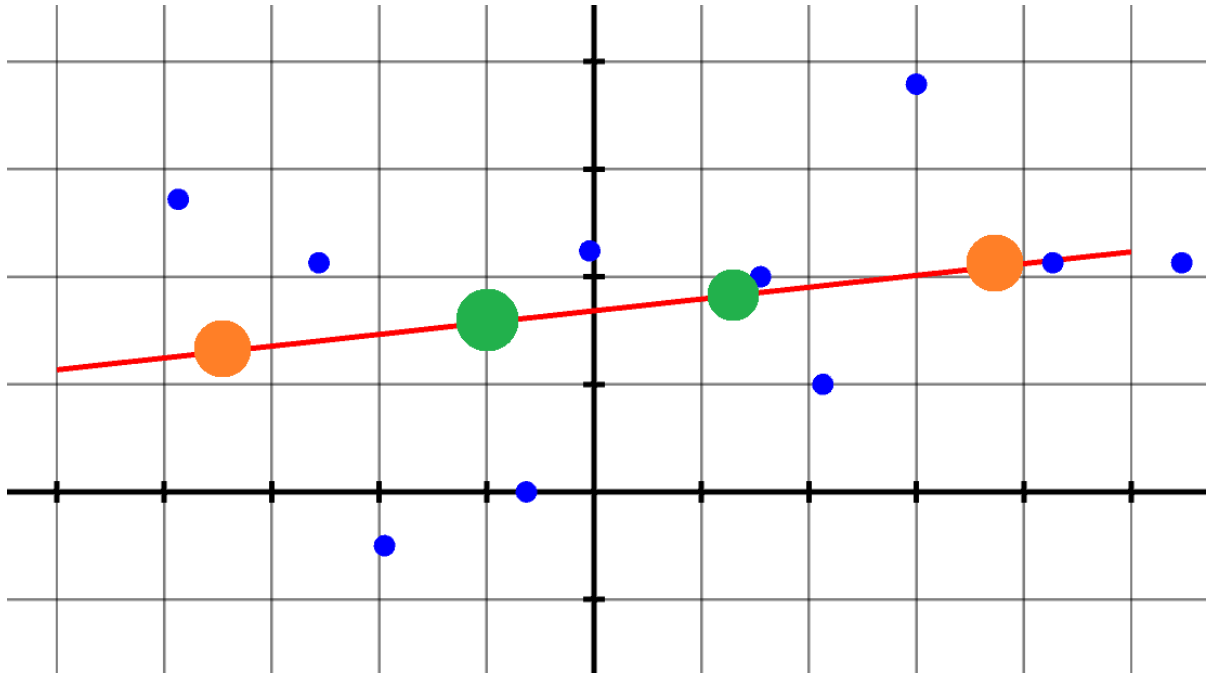
Copyright © by Wojtek Sterna

8.8. Least Squares Method

- One of the applications is the DXT image compression algorithm:



https://commons.wikimedia.org/wiki/File:RGB_color_cube.svg Author User:Maklaan



Copyright © by Wojtek Sterna

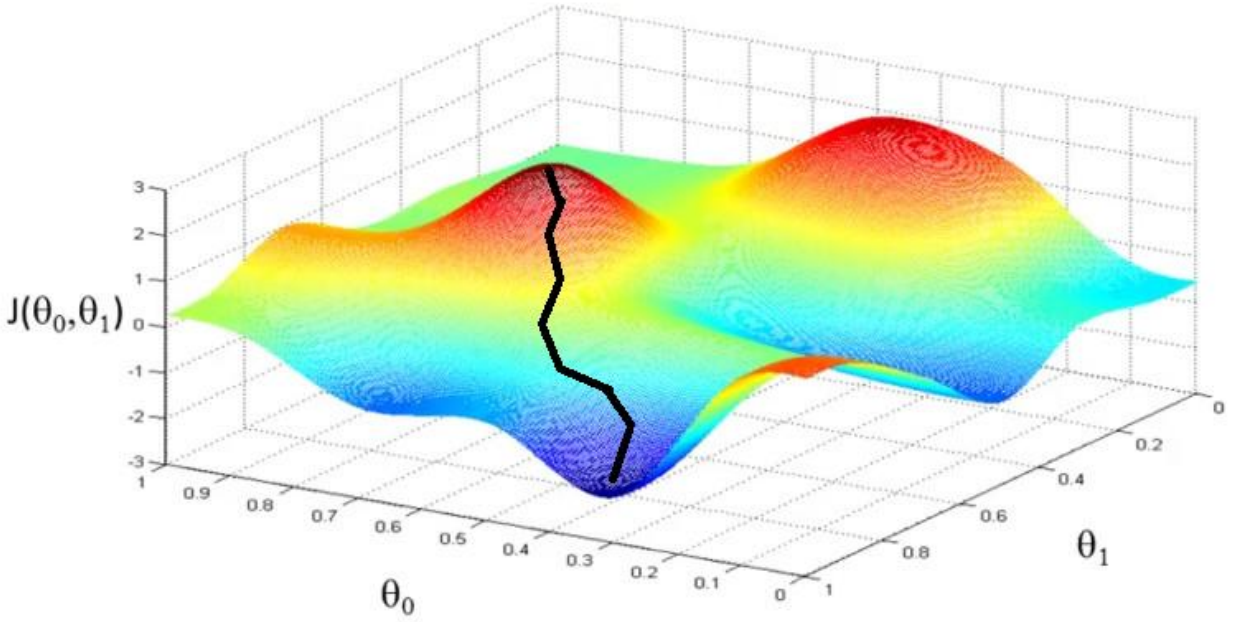
8.9. Gradient Descent Algorithm

- In the least squares method we solve equations and determine the values of parameters under the condition that the partial derivatives are 0
- Sometimes those equations can be very laborious. So much that it becomes very difficult or even impossible to solve them
- If that is the case we can use some numerical/approximate algorithms to solve those equations. One such algorithm is the so-called **gradient descent**

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm

- Let's say we have a function of two variables $f(a, b)$. It may be the cost function we saw earlier, which tells the sum of the (squared) distances of points from a straight line
- If we calculate the gradient of that function at a certain point (a, b), we will get the direction along which we can go to obtain a **larger** value of f



https://www.internalpointers.com/post/gradient-descent-function

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm

- However, if we go in the opposite direction we will obtain a **smaller** value of f
- By going in the opposite direction long enough we are likely to reach a minimum of the function
- We need to pick a starting point for the search. In theory we can pick any point, but the closer one we choose to the solution the better
- https://www.internalpointers.com/post/gradient-descent-function

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Line Equation

- Let's take the cost function from the previous problem (of finding the straight line that best approximates a set of points):

$$D(a,b) = \sum_{i=1}^{n} ((ax_i + b) - y_i)^2$$

```
float Cost(Vector2[] pts, float a, float b)
{
    float cost = 0.0f;
    for (int i = 0; i < pts.Length; i++)
    {
        float value = (a * pts[i].x + b) - pts[i].y;
        cost += value * value;
    }

    return cost;
}
```

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Line Equation

- We now calculate (numerically) the gradient of that cost function and modify the variables, **iterating** a specified number of times at a given **step**:

```
const float h = 0.01f;
const float gradientStep = 0.01f;

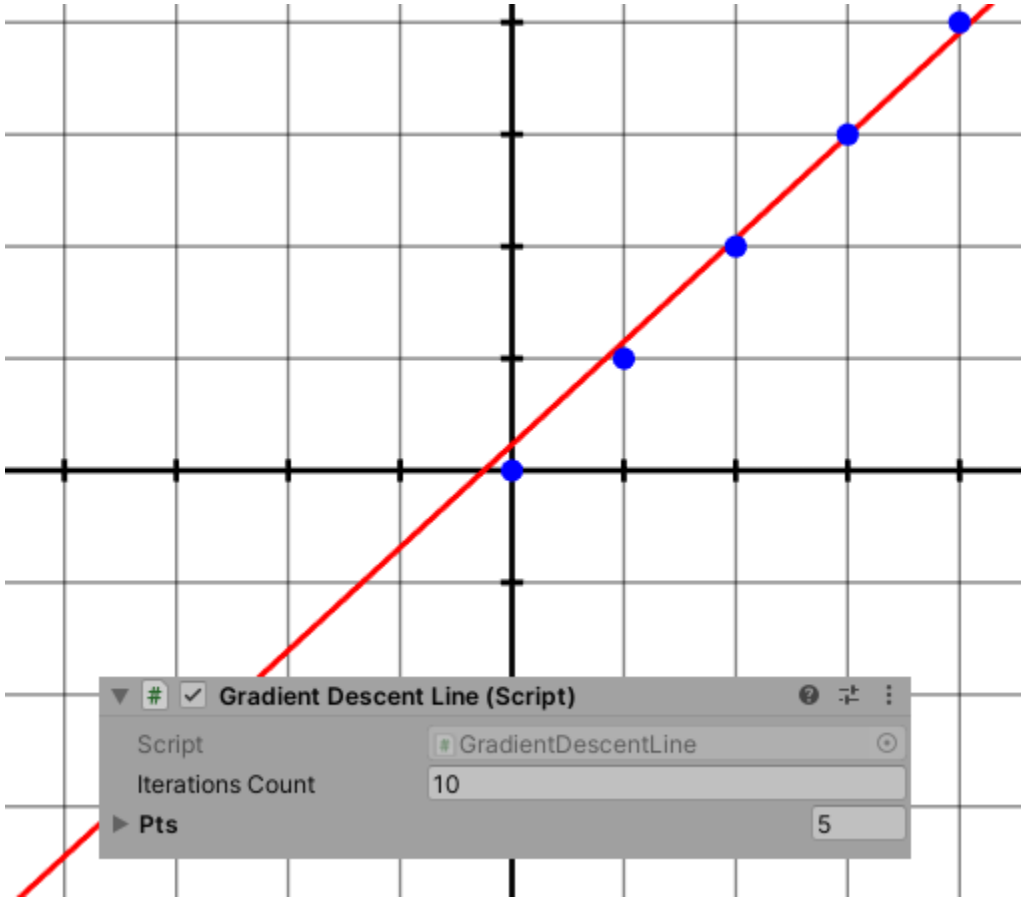
for (int i = 0; i < iterationsCount; i++)
{
    float dCost_dA = ( Cost(pts, a + h, b) - Cost(pts, a - h, b) ) / ( 2.0f * h );
    float dCost_dB = ( Cost(pts, a, b + h) - Cost(pts, a, b - h) ) / ( 2.0f * h );

    a -= gradientStep * dCost_dA;
    b -= gradientStep * dCost_dB;

    Debug.Log("i = " + i + ", " + "cost = " + Cost(pts, a, b));
}
```

Copyright © by Wojtek Sterna

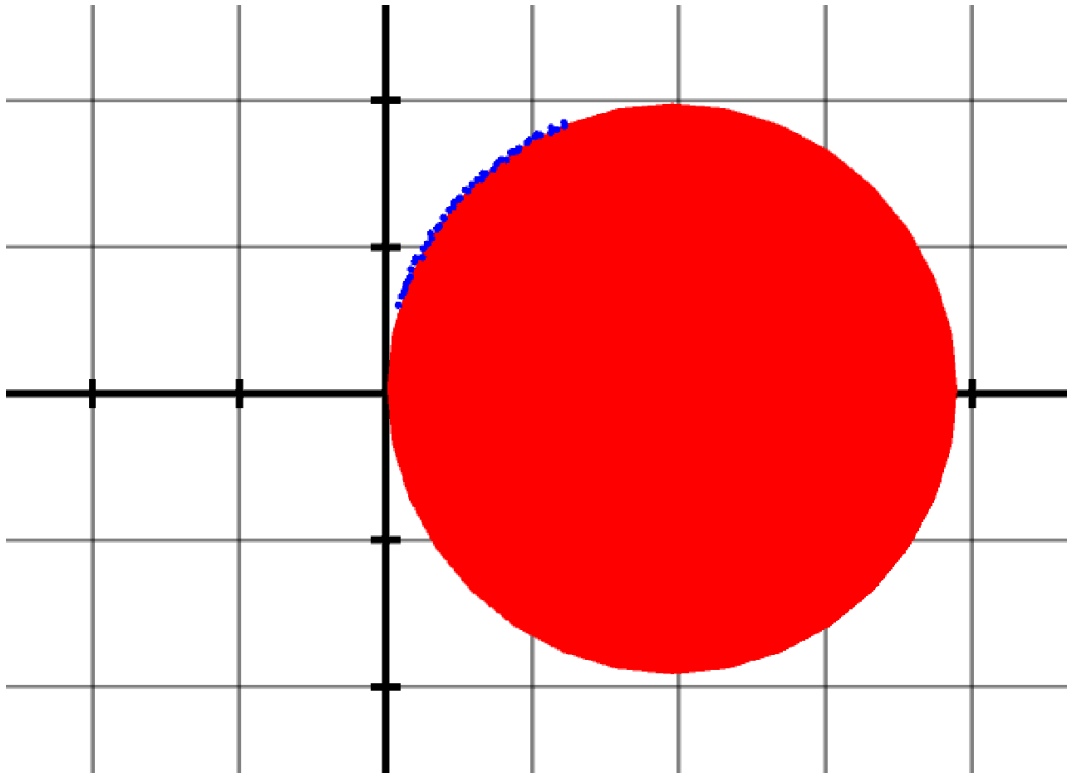
8.9. Gradient Descent Algorithm – Line Equation



Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation

- We will now use gradient descent to find the circle equation that best approximates a certain set of points:



Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation

- The circle equation:

$$(x-a)^2+(y-b)^2-r^2=0$$

- The cost function for a single point:

$$D(a,b,r)=\left((x-a)^2+(y-b)^2-r^2\right)^2$$

- The cost function for a set of points:

$$D(a,b,r)=\sum_{i=1}^{n}\left((x_i-a)^2+(y_i-b)^2-r^2\right)^2$$

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation

- The cost function in source code:

```
private float CostABR(Vector2[] pts, float a, float b, float r)
{
    float cost = 0.0f;
    for (int i = 0; i < pts.Length; i++)
    {
        float circleEq = Sqr(pts[i].x - a) + Sqr(pts[i].y - b) - Sqr(r);
        cost += circleEq * circleEq;
    }

    return cost;
}
```

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation

- We now calculate the gradient and go in its opposite direction:

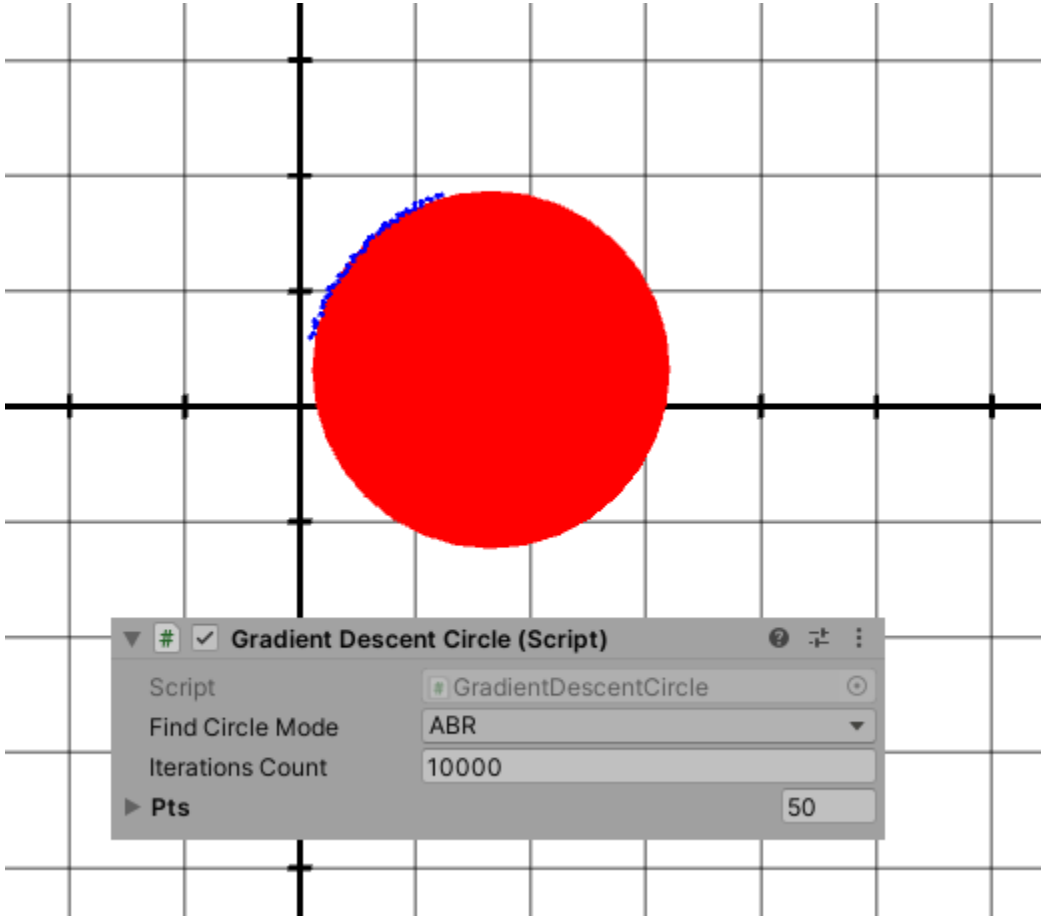
```
for (int i = 0; i < iterationsCount; i++)
{
    float dCost_dA = ( CostABR(pts, circlePos.x + h, circlePos.y, circleR) - CostABR(pts, circlePos.x - h, circlePos.y, circleR) ) / ( 2.0f * h );
    float dCost_dB = ( CostABR(pts, circlePos.x, circlePos.y + h, circleR) - CostABR(pts, circlePos.x, circlePos.y - h, circleR) ) / ( 2.0f * h );
    float dCost_dR = ( CostABR(pts, circlePos.x, circlePos.y, circleR + h) - CostABR(pts, circlePos.x, circlePos.y, circleR - h) ) / ( 2.0f * h );
    dCost_dA /= pts.Length;
    dCost_dB /= pts.Length;
    dCost_dR /= pts.Length;

    circlePos.x -= gradientStep * dCost_dA;
    circlePos.y -= gradientStep * dCost_dB;
    circleR -= gradientStep * dCost_dR;
}
```

- Note the division/normalization! In the previous example we should have also taken care of it

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation



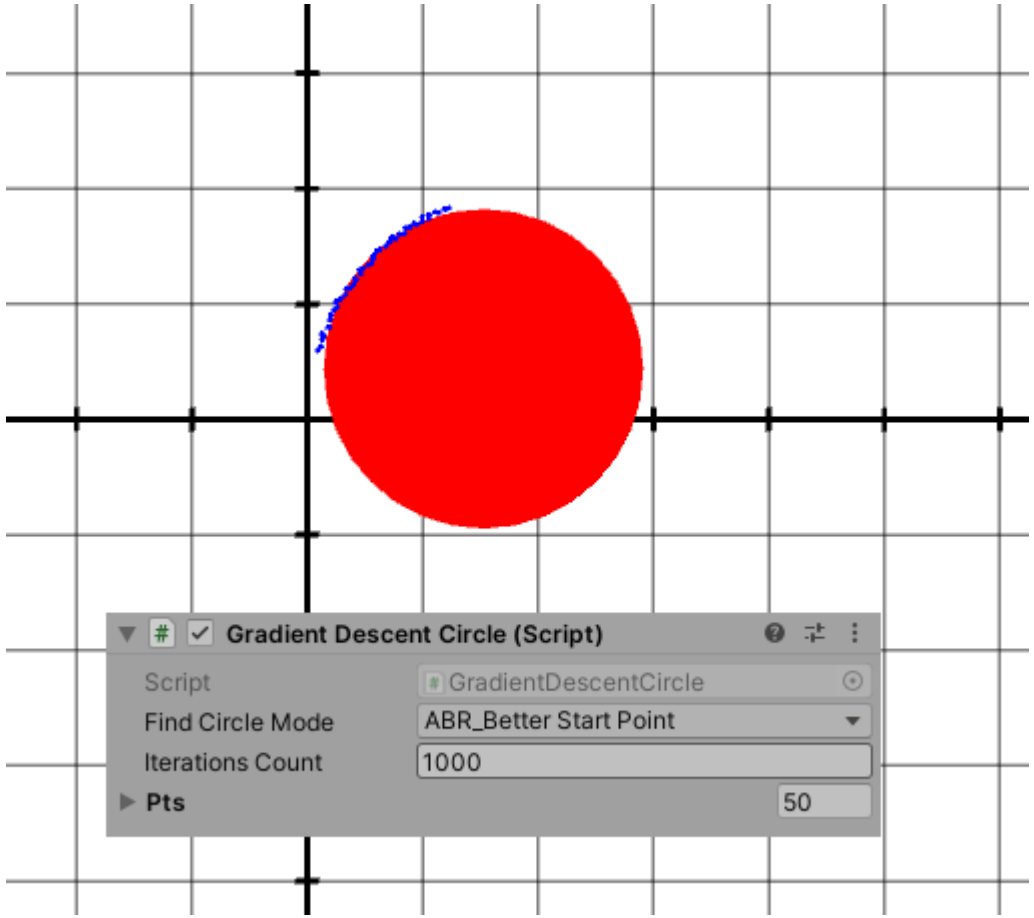
Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation

- The better the starting point the faster we will reach a better result

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation



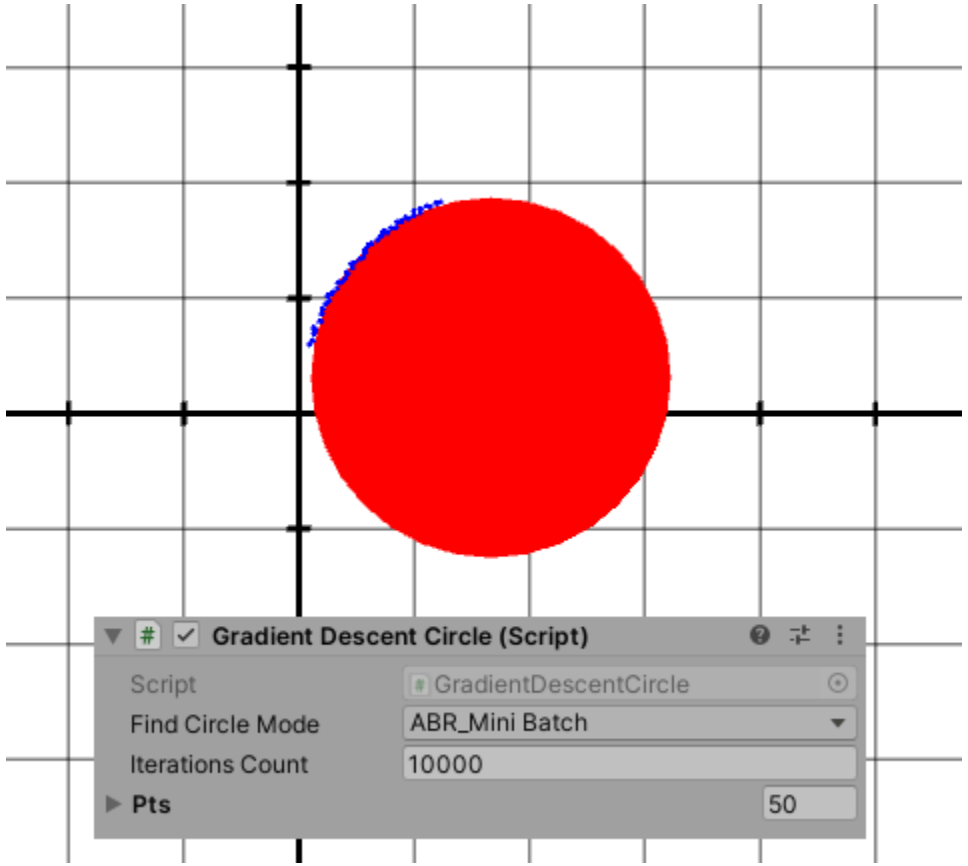
Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation

- We do not need to every time go through all of the points. For each iteration we can use a different subset (batch) of points

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation



Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Circle Equation

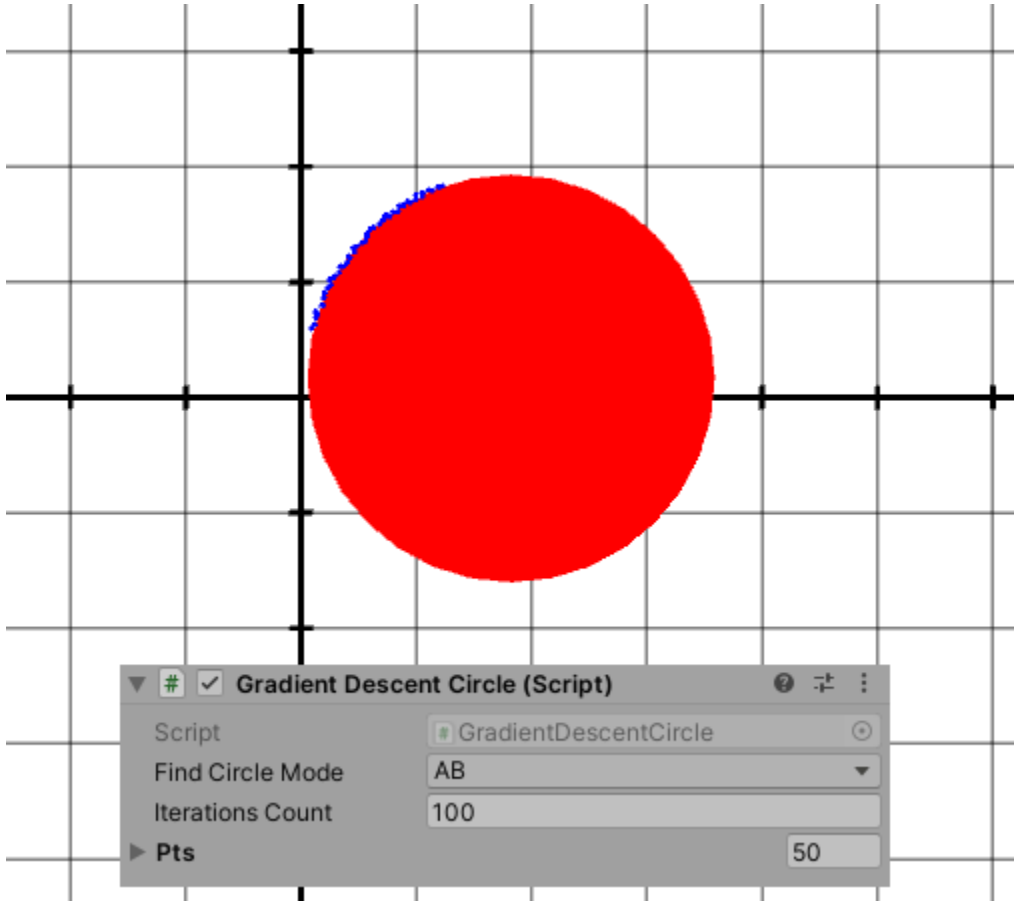
- In the example being discussed we can simplify the problem a bit
- We can get rid off of the radius in the cost function:

$$D(a,b) = \sum_{i=1}^{n}((x_i - a)^2 + (y_i - b)^2 - r^2)^2$$

- The radius is obviously still needed there in the cost function. We will however calculate it „automatically” based on the distances between the current circle’s origin (a, b) and the points

Copyright © by Wojtek Sterna

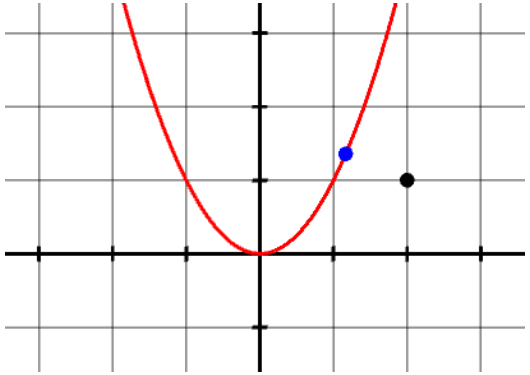
8.9. Gradient Descent Algorithm – Circle Equation



Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Distance Between Point and Parabola

- Previously, we wrote the program ExtremaParabolaPointDistance, which calculated a point on a parabola that was closest to another selected point:



- The analytic solution was cumbersome:
  1) it was lengthy and involving
  2) calculations required the use of complex numbers

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Distance Between Point and Parabola

- Like every problem that we solve with the gradient descent algorithm, we start with the cost function that we need to minimize:

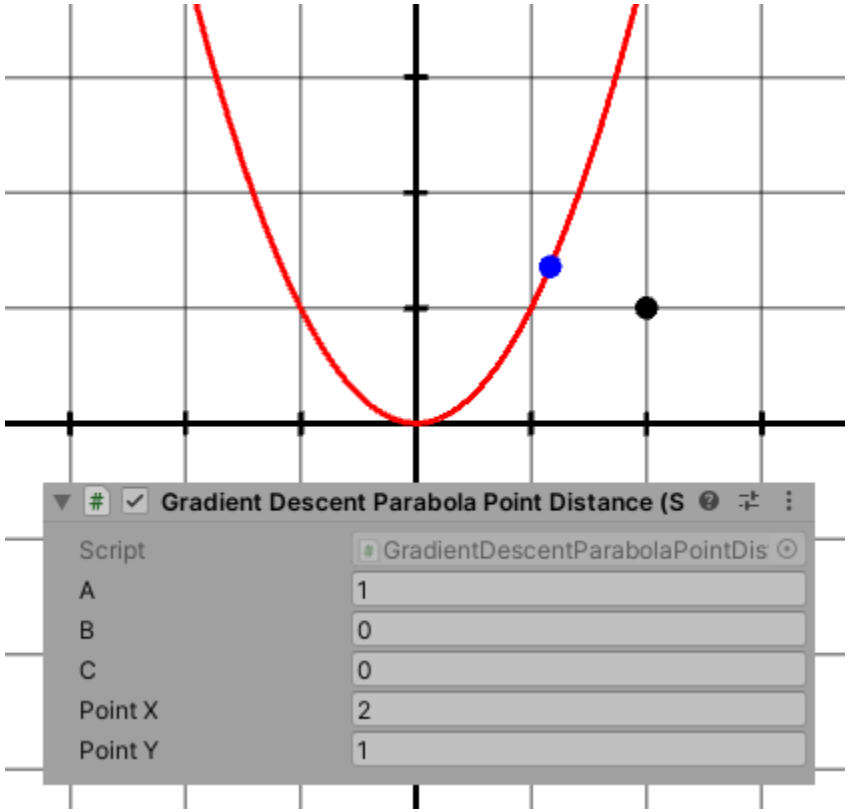
$$D(x) = (x - p_x)^2 + \Big((ax^2 + bx + c\,) - p_y\Big)^2$$

- First we need the function's derivative (which by the way can also be calculated numerically). We solved it before (analytically) when we talked about the ExtremaParabolaPointDistance program:

$$D'(x) = 2\Big((2ax + b)\big(ax^2 + bx + c - p_y\big) + x - p_x\Big)$$

Copyright © by Wojtek Sterna

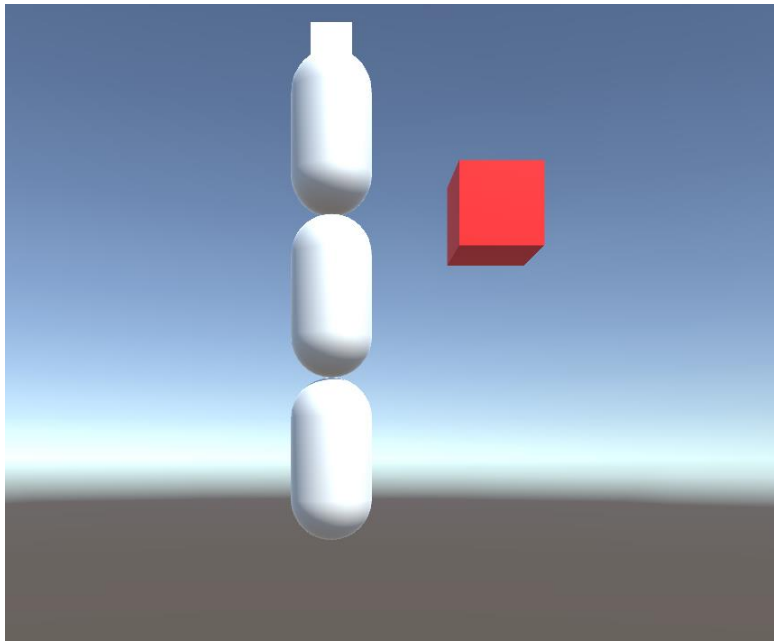
8.9. Gradient Descent Algorithm – Distance Between Point and Parabola

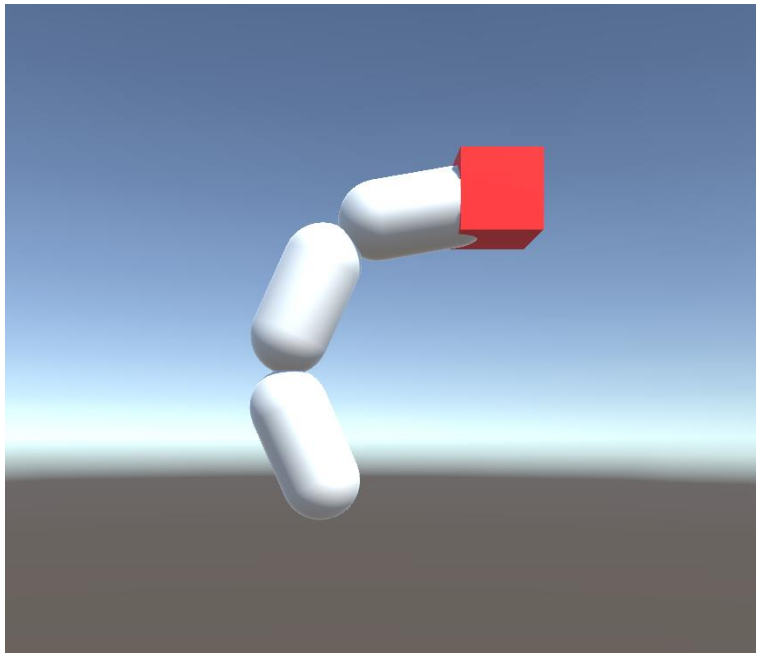


Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Inverse Kinematics

- One example that demonstrates the wide use of the GD algorithm is the **inverse kinematics** problem:





Copyright © by Wojtek Sterna

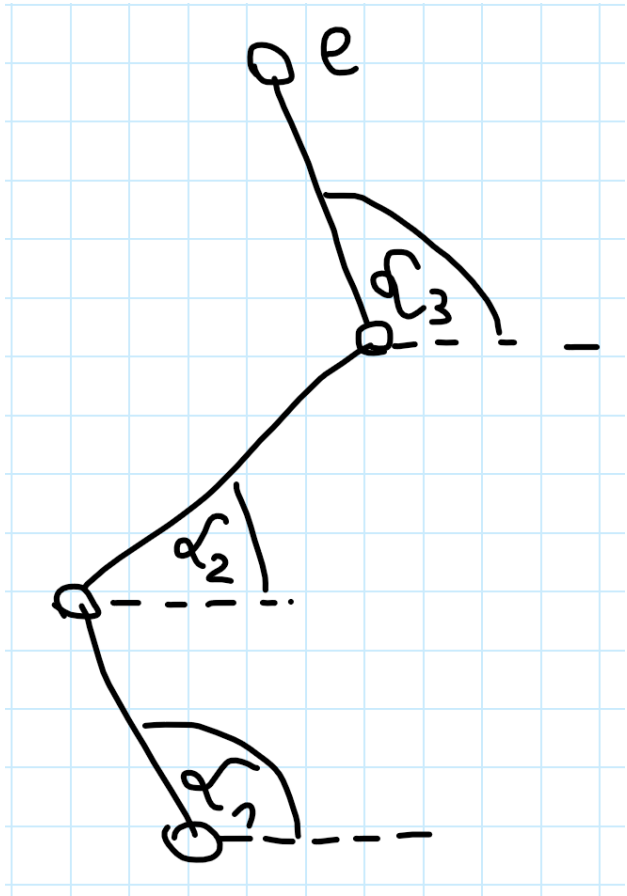
8.9. Gradient Descent Algorithm – Inverse Kinematics

- In **forward kinematics** on input we have **geometric parameters**, such as angles between objects/joints and based on that we calculate the position of the so-called **end effector**
- In **inverse kinematics** it is the opposite: on input we give the position of the end effector and we want to find the values of the geometric parameters

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Inverse Kinematics

- Three geometric parameters α_1, α_2, α_3 and the end effector e:



Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Inverse Kinematics

- Forward kinematics:

$$f(\alpha_1, \alpha_2, \dots, \alpha_n) = e \qquad e \ - \text{end effector}$$

- Inverse kinematics:

$$g(e) = [\alpha_1, \alpha_2, \dots, \alpha_n] \qquad e \ - \text{end effector target}$$

- We know the f function
- Our goal is to find the g function, which is simply the inverse of f. For given e it should return a chain of parameters $[\alpha_1, \alpha_2, \dots, \alpha_n]$

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Inverse Kinematics

- Various sets of geometric parameters can lead to the same position of the end effector
- Which set will be returned by the g function is not always obvious
- It can be influenced by **constraints**, which we might need to be met in an IK solution

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Inverse Kinematics

- There are a few algorithms that we can solve the IK problem with (like FABRIK or Cyclic Coordinate Descent)
- One of them is the gradient descent
- We start off with some initial angles between joints, which **do not yet** give us the desired end effector e position. Instead, they give us „some initial" location e_0:

$$f(\alpha_1, \alpha_2, \dots, \alpha_n) = e_0$$

- Expression $|e - e_0|$ is the distance between the desired end effector e and e_0. That distance is the cost that we want to minimize
- In practice we will of course want to minimize $(e - e_0)^2$

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Inverse Kinematics

2 references
```
private Vector3 F(float angle1, float angle2, float angle3)
{
    float x1 = 0.0f;
    float y1 = 2.0f;
    float x2 = 0.0f;
    float y2 = 4.0f;
    float x3 = 0.0f;
    float y3 = 6.0f;

    RotateAround(x1, y1, angle1, 0.0f, 0.0f, out x1, out y1);
    RotateAround(x2, y2, angle1, 0.0f, 0.0f, out x2, out y2);
    RotateAround(x3, y3, angle1, 0.0f, 0.0f, out x3, out y3);

    RotateAround(x2, y2, angle2, x1, y1, out x2, out y2);
    RotateAround(x3, y3, angle2, x1, y1, out x3, out y3);

    RotateAround(x3, y3, angle3, x2, y2, out x3, out y3);

    return new Vector3(x3, y3, 0.0f);
}
```

6 references
```
private void RotateAround(
    float x, float y, float angle,
    float rx, float ry, out float x2, out float y2)
{
    x -= rx;
    y -= ry;
    Rotate(x, y, angle, out x2, out y2);
    x2 += rx;
    y2 += ry;
}
```

1 reference
```
private void Rotate(float x, float y, float angle, out float x2, out float y2)
{
    angle *= Mathf.Deg2Rad;
    x2 = x * Mathf.Cos(angle) - y * Mathf.Sin(angle);
    y2 = x * Mathf.Sin(angle) + y * Mathf.Cos(angle);
}
```

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Inverse Kinematics

```
1 reference
private void G(ref float angle1, ref float angle2, ref float angle3, Vector3 endEffectorTargetPosition)
{
    const float gradientStep = 10.0f;
    const float h = 0.001f;

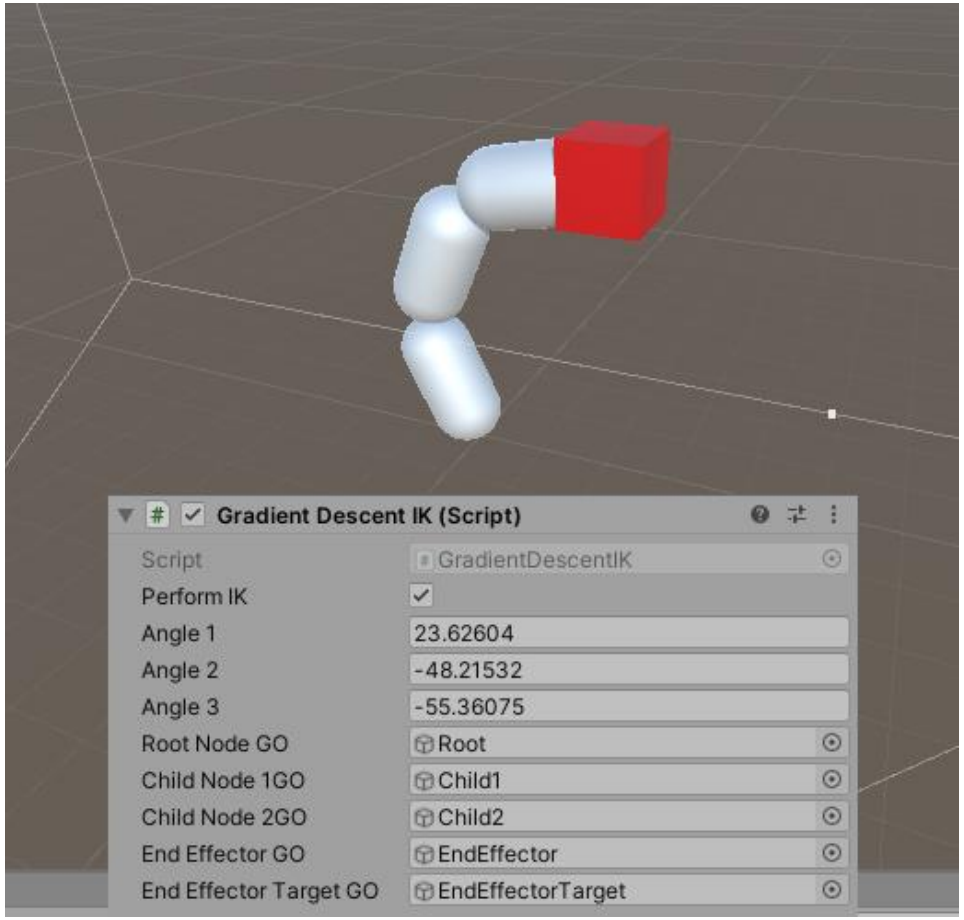
    for (int i = 0; i < 10; i++)
    {
        float dCost_dAngle1 = ( Cost(angle1 + h, angle2, angle3, endEffectorTargetPosition) - Cost(angle1 - h, angle2, angle3, endEffectorTargetPosition) ) / ( 2.0f * h );
        float dCost_dAngle2 = ( Cost(angle1, angle2 + h, angle3, endEffectorTargetPosition) - Cost(angle1, angle2 - h, angle3, endEffectorTargetPosition) ) / ( 2.0f * h );
        float dCost_dAngle3 = ( Cost(angle1, angle2, angle3 + h, endEffectorTargetPosition) - Cost(angle1, angle2, angle3 - h, endEffectorTargetPosition) ) / ( 2.0f * h );

        angle1 -= gradientStep * dCost_dAngle1;
        angle2 -= gradientStep * dCost_dAngle2;
        angle3 -= gradientStep * dCost_dAngle3;
    }
}

6 references
private float Cost(float angle1, float angle2, float angle3, Vector3 endEffectorTargetPosition)
{
    Vector3 endEffectorCurrentPosition = F(angle1, angle2, angle3);
    return Vector3.SqrMagnitude(endEffectorCurrentPosition - endEffectorTargetPosition);
}
```

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Inverse Kinematics



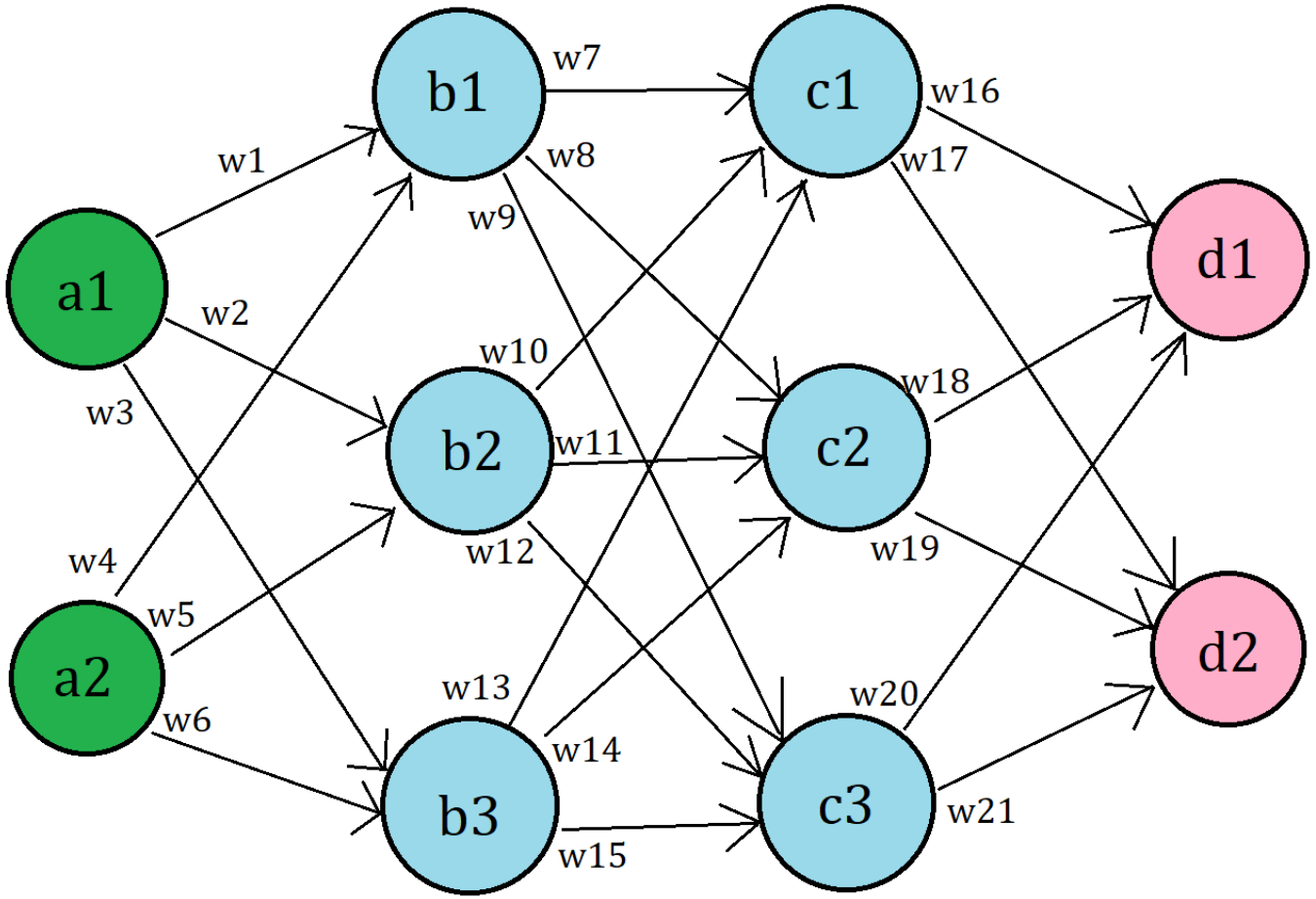
Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Neural Nets

- The GD algorithm is the foundation of **neural networks**
- The idea of neural networks is to construct a cost function for thousands, millions or even billions of parameters and their „gradient descending” for as long until we’ve found optimal values for those parameters

Copyright © by Wojtek Sterna

8.9. Gradient Descent Algorithm – Neural Nets




```
1   F(a1, a2) = [d1, d2]
2   {
3       b1 = a1*w1 + a2*w4;
4       b2 = a1*w2 + a2*w5;
5       b3 = ...
6       c1 = b1*w7 + b2*w10 + b3*w13;
7       c2 = ...
8       c3 = ...
9
10      d1 = c1*w16 + c2*w18 + c3*w20;
11      d2 = c1*w17 + c2*w19 + c3*w21;
12  }
```

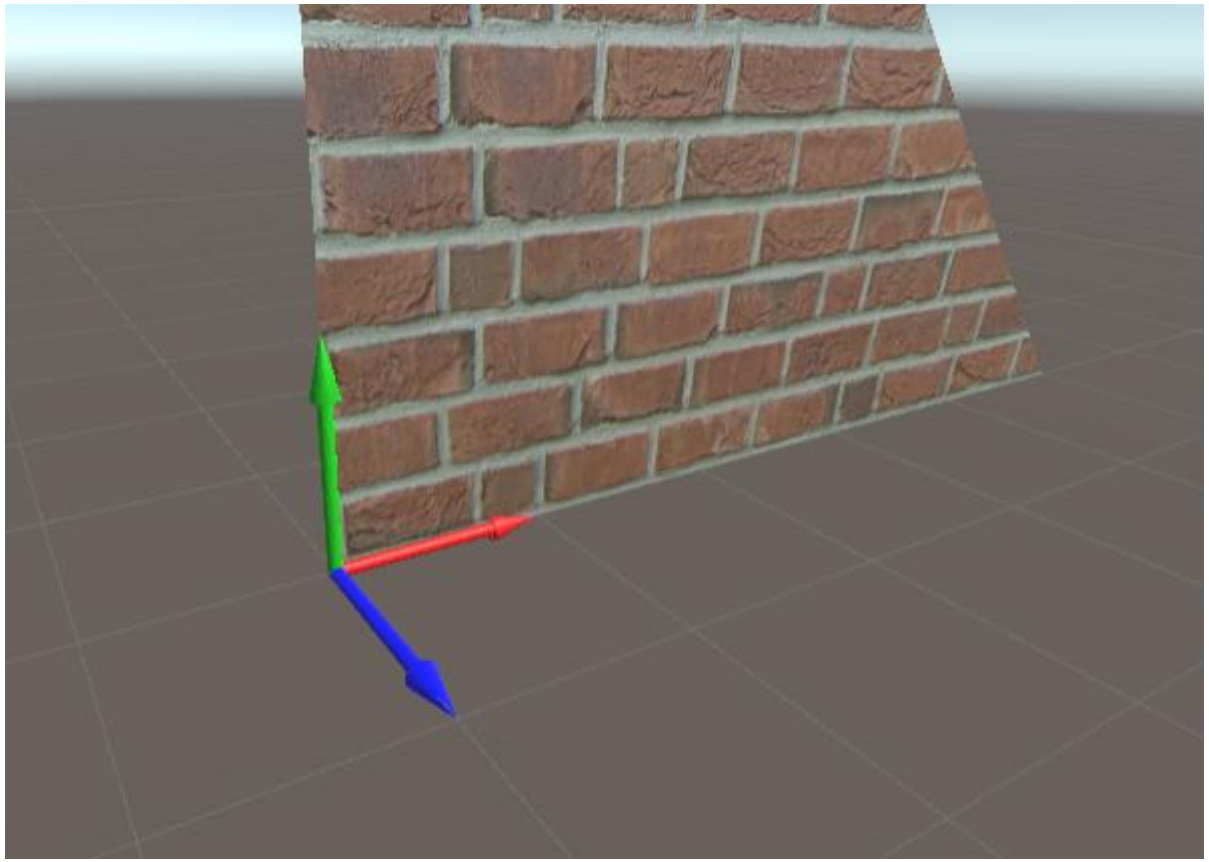
```
1   Cost(w1, ..., w21) = cost
2   {
3       cost = 0.0f;
4
5       foreach (entry in dataset)
6       {
7           [d1, d2] = F(entry.a1, entry.a2);
8
9           cost +=
10              sqr(entry.d1 - d1) +
11              sqr(entry.d2 - d2);
12      }
13  }
```

$$c1 = b1*w7 + b2*w10 + b3*w13$$

Copyright © by Wojtek Sterna

8.10. Tangent Space

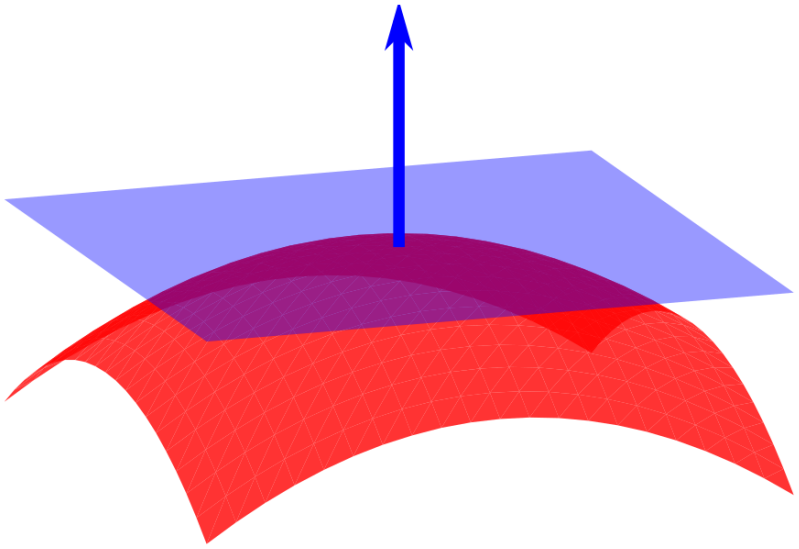
- **Tangent space** is three basis vectors that define the local space of a vertex or a triangle/surface:



Copyright © by Wojtek Sterna

8.10. Tangent Space

- One of those vectors is the well-known normal vector $\vec{n}$
- The remaining two vectors are **tangent vectors** $\vec{t}$ and $\vec{b}$
- The tangent vectors define a **tangent plane** that the normal vector is perpendicular to:



https://en.wikipedia.org/wiki/File:Surface_normal_illustration.svg **Author** Patrick87

Copyright © by Wojtek Sterna

8.10. Tangent Space

- Overall tangent vectors can have arbitrary directions within the tangent plane, but usually for practical reasons we want them to be perpendicular and have them form – together with the normal vector – an orthogonal basis:

$$\vec{t} \times \vec{b} = \vec{n}$$

$$\vec{b} \times \vec{n} = \vec{t}$$

$$\vec{n} \times \vec{t} = \vec{b}$$

Copyright © by Wojtek Sterna

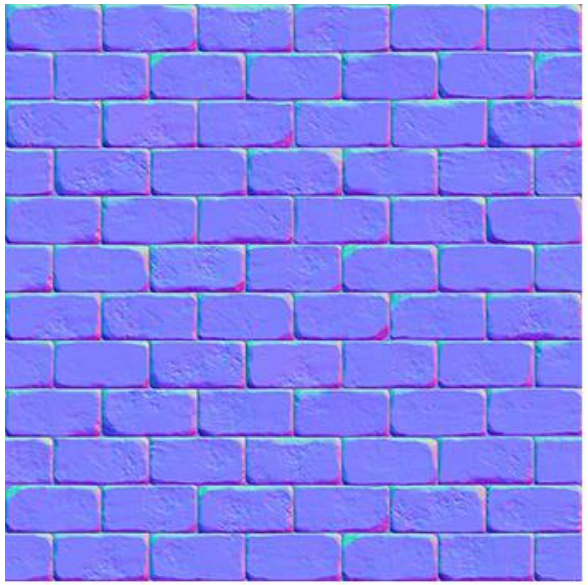
8.10. Tangent Space

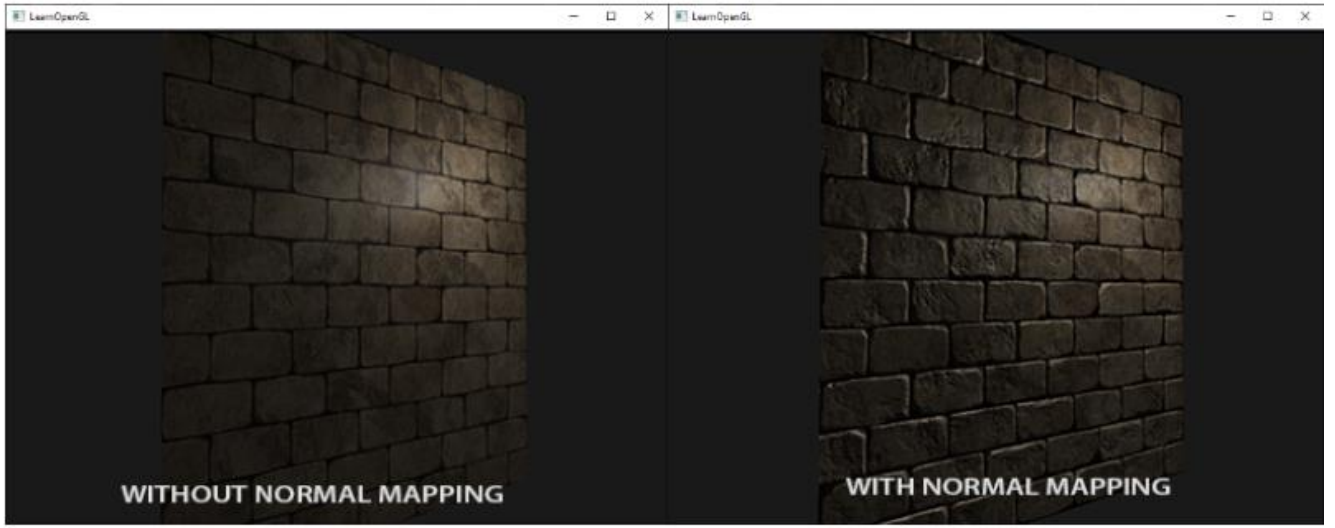
- In 3D engines almost always the tangent vectors have very specific directions. They are dictated by the texture coordinates of the vertices
- The tangent vectors point in the directions along which the vertices' texture coordinates (u, v) increase

Copyright © by Wojtek Sterna

8.10. Tangent Space

- When tangent vectors are determined in this fashion they enable effects like **normal mapping** and alike
- https://learnopengl.com/Advanced-Lighting/Normal-Mapping:





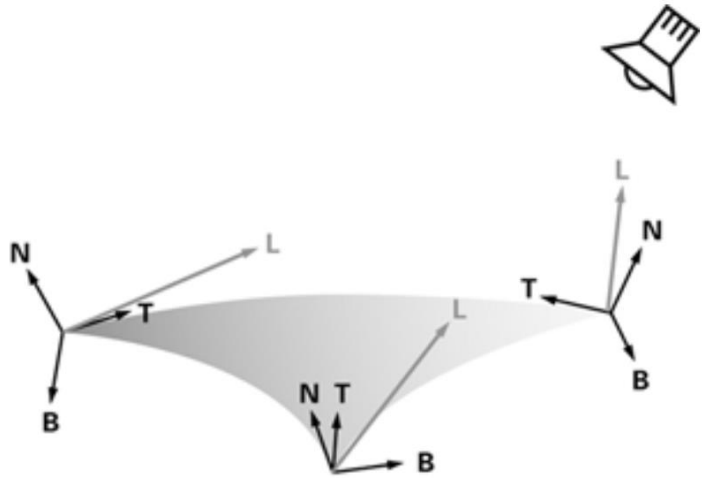
https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Copyright © by Wojtek Sterna

8.10. Tangent Space

- **Normal maps** are usually encoded in the tangent space
- Color value $(0, 128, 255)$ corresponds to vector $[-1, 0, 1]$
- When calculating a pixel's lighting we can either transform the normal vector from the tangent space to e.g. world space, or we can transform the light direction from world to tangent space

$\vec{N} \circ \vec{L}$



https://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter08.html

Copyright © by Wojtek Sterna

8.10. Tangent Space

- Let's say that we have vectors $\vec{t}$, $\vec{b}$ and $\vec{n}$ which describe a tangent space and those vectors are expressed in world space.
  Then the matrix T is a matrix that transforms vectors from the tangent space to world space:

$$T = \begin{bmatrix} \vec{t}_x & \vec{b}_x & \vec{n}_x \\ \vec{t}_y & \vec{b}_y & \vec{n}_y \\ \vec{t}_z & \vec{b}_z & \vec{n}_z \end{bmatrix} \qquad \begin{bmatrix} \vec{t}_x & \vec{b}_x & \vec{n}_x \\ \vec{t}_y & \vec{b}_y & \vec{n}_y \\ \vec{t}_z & \vec{b}_z & \vec{n}_z \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \vec{n}_x \\ \vec{n}_y \\ \vec{n}_z \end{bmatrix}$$

- If T is **orthonormal** then its transpose is also its inverse (which permits fast inverse transform)

Copyright © by Wojtek Sterna

8.10. Tangent Space

- We will now derive the formula for calculating $\vec{t}$ and $\vec{b}$ for any triangle
- On input we have a triangle's vertices (x_i, y_i, z_i) and texture coordinates (u_i, v_i), $i = 1, 2, 3$
- Note that the texture coordinates can be thought of as parameters in the parametric plane equation:

$$p(u, v) = o + u\vec{t} + v\vec{b}$$

$$p(u_i, v_i) = (x_i, y_i, z_i)$$

Copyright © by Wojtek Sterna

8.10. Tangent Space

- Let’s state the facts as separate equations:

$$\begin{cases} \boldsymbol{o} + u_1\vec{\boldsymbol{t}} + v_1\vec{\boldsymbol{b}} = (x_1, y_1, z_1) \\ \boldsymbol{o} + u_2\vec{\boldsymbol{t}} + v_2\vec{\boldsymbol{b}} = (x_2, y_2, z_2) \\ \boldsymbol{o} + u_3\vec{\boldsymbol{t}} + v_3\vec{\boldsymbol{b}} = (x_3, y_3, z_3) \end{cases}$$

- These are 3 equations with 3 unknowns
- Or actually 9 equations with 9 unknowns (after expanding)

Copyright © by Wojtek Sterna

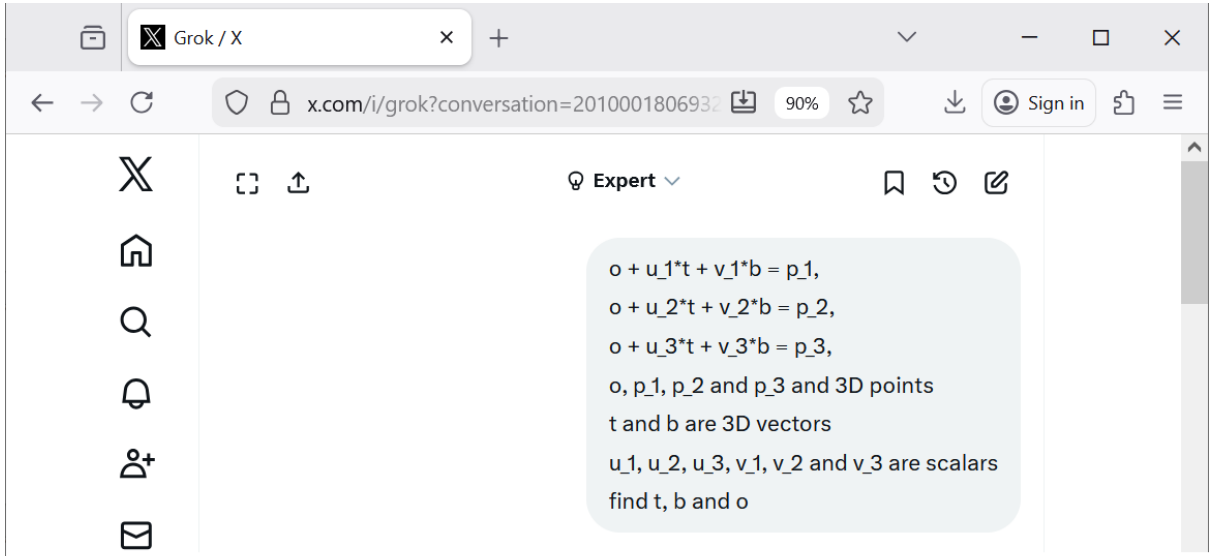
8.10. Tangent Space

- After expanding:

$$\begin{cases} \boldsymbol{o_x} + u_1\vec{\boldsymbol{t}}_{\boldsymbol{x}} + v_1\vec{\boldsymbol{b}}_{\boldsymbol{x}} = x_1 \\ \boldsymbol{o_y} + u_1\vec{\boldsymbol{t}}_{\boldsymbol{y}} + v_1\vec{\boldsymbol{b}}_{\boldsymbol{y}} = y_1 \\ \boldsymbol{o_z} + u_1\vec{\boldsymbol{t}}_{\boldsymbol{z}} + v_1\vec{\boldsymbol{b}}_{\boldsymbol{z}} = z_1 \\ \dots \\ \boldsymbol{o_z} + u_3\vec{\boldsymbol{t}}_{\boldsymbol{z}} + v_3\vec{\boldsymbol{b}}_{\boldsymbol{z}} = z_3 \end{cases}$$

Copyright © by Wojtek Sterna

8.10. Tangent Space



The explicit expressions (obtained by computing A^{-1} symbolically) are:

$$\mathbf{o} = \frac{u_2 v_3 - u_3 v_2}{\Delta}\mathbf{p_1} + \frac{-u_1 v_3 + u_3 v_1}{\Delta}\mathbf{p_2} + \frac{u_1 v_2 - u_2 v_1}{\Delta}\mathbf{p_3}$$

$$\mathbf{t} = \frac{v_2 - v_3}{\Delta}\mathbf{p_1} + \frac{-v_1 + v_3}{\Delta}\mathbf{p_2} + \frac{v_1 - v_2}{\Delta}\mathbf{p_3}$$

$$\mathbf{b} = \frac{-u_2 + u_3}{\Delta}\mathbf{p_1} + \frac{u_1 - u_3}{\Delta}\mathbf{p_2} + \frac{-u_1 + u_2}{\Delta}\mathbf{p_3}$$

Copyright © by Wojtek Sterna

8.10. Tangent Space

- Let's try another derivation approach (a more geometric one)
- Plane equation split for each component:

$$\begin{cases} x(u,v) = o_x + u\vec{t}_x + v\vec{b}_x \\ y(u,v) = o_y + u\vec{t}_y + v\vec{b}_y \\ z(u,v) = o_z + u\vec{t}_z + v\vec{b}_z \end{cases}$$

- Now let's calculate partial derivatives of $x(u,v)$:

$$\frac{\partial x}{\partial u} = \vec{t}_x \qquad \frac{\partial x}{\partial v} = \vec{b}_x$$

Copyright © by Wojtek Sterna

8.10. Tangent Space

- All partial derivatives:

$$\left[\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}\right] = [\vec{t}_x, \vec{t}_y, \vec{t}_z] \qquad \left[\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v}\right] = [\vec{b}_x, \vec{b}_y, \vec{b}_z]$$

- We've just shown that tangent vectors can be obtained by calculating partial derivatives with respect to u and v (but we know this already from the sub-chapter about parametric function derivative)
- It also means that if we somehow calculate those partial derivatives in such a way that in the results only (x_i, y_i, z_i) and (u_i, v_i) occur, then we will have calculated tangent vectors
- Tangent vectors calculated in this way will have directions along increasing texture coordinates

Copyright © by Wojtek Sterna

8.10. Tangent Space

- Each vertex is described by five values $(x_i, y_i, z_i, u_i, v_i)$
- All those values change linearly within the single plane on which the triangle lies
- We can for example write the following implicit equation of the plane (on which the triangle lies):

$$Ax + By + Cz + D = 0$$

Copyright © by Wojtek Sterna

8.10. Tangent Space

- Nothing stands in our way though to write the following implicit plane equation:

$$A\boldsymbol{x} + B\boldsymbol{u} + C\boldsymbol{v} + D = 0$$

- Coefficients A, B, C and D can be calculated using standard methods
- Let's solve for x:

$$x = -\frac{Bu + Cv + D}{A}$$

$$x(u, v) = -\frac{Bu + Cv + D}{A}$$

Copyright © by Wojtek Sterna

8.10. Tangent Space

- The calculated x is actually a function of two variables (we've seen it before):

$$x(u,v) = -\frac{Bu + Cv + D}{A}$$

- Let's calculate its partial derivatives:

$$\frac{\partial x}{\partial u} = -\frac{B}{A} \qquad \frac{\partial x}{\partial v} = -\frac{C}{A}$$

- We've just calculated $\vec{t}_x$ and $\vec{b}_x$

Copyright © by Wojtek Sterna

8.10. Tangent Space

- We can do the same for y and z. Thus in total we have three plane equations:

$$A_1\boldsymbol{x} + B_1\boldsymbol{u} + C_1\boldsymbol{v} + D_1 = 0$$

$$A_2\boldsymbol{y} + B_2\boldsymbol{u} + C_2\boldsymbol{v} + D_2 = 0$$

$$A_3\boldsymbol{z} + B_3\boldsymbol{u} + C_3\boldsymbol{v} + D_3 = 0$$

Copyright © by Wojtek Sterna

8.10. Tangent Space

- In the end:

$$\frac{\partial x}{\partial u} = \vec{t}_x = -\frac{B_1}{A_1} \qquad \frac{\partial x}{\partial v} = \vec{b}_x = -\frac{C_1}{A_1}$$

$$\frac{\partial y}{\partial u} = \vec{t}_y = -\frac{B_2}{A_2} \qquad \frac{\partial y}{\partial v} = \vec{b}_y = -\frac{C_2}{A_2}$$

$$\frac{\partial z}{\partial u} = \vec{t}_z = -\frac{B_3}{A_3} \qquad \frac{\partial z}{\partial v} = \vec{b}_z = -\frac{C_3}{A_3}$$

- More concisely:

$$\vec{t} = \left[-\frac{B_1}{A_1}, -\frac{B_2}{A_2}, -\frac{B_3}{A_3}\right] \qquad \vec{b} = \left[-\frac{C_1}{A_1}, -\frac{C_2}{A_2}, -\frac{C_3}{A_3}\right]$$

Copyright © by Wojtek Sterna

8.10. Tangent Space

```
Vector4 xuvPlane = GetPlaneEquation(new Vector3(v1.x, uv1.x, uv1.y), new Vector3(v2.x, uv2.x, uv2.y), new Vector3(v3.x, uv3.x, uv3.y));
Vector4 yuvPlane = GetPlaneEquation(new Vector3(v1.y, uv1.x, uv1.y), new Vector3(v2.y, uv2.x, uv2.y), new Vector3(v3.y, uv3.x, uv3.y));
Vector4 zuvPlane = GetPlaneEquation(new Vector3(v1.z, uv1.x, uv1.y), new Vector3(v2.z, uv2.x, uv2.y), new Vector3(v3.z, uv3.x, uv3.y));

Vector3 tangent;
tangent.x = -xuvPlane.y / xuvPlane.x;
tangent.y = -yuvPlane.y / yuvPlane.x;
tangent.z = -zuvPlane.y / zuvPlane.x;
tangent.Normalize();

Vector3 bitangent;
bitangent.x = -xuvPlane.z / xuvPlane.x;
bitangent.y = -yuvPlane.z / yuvPlane.x;
bitangent.z = -zuvPlane.z / zuvPlane.x;
bitangent.Normalize();
```

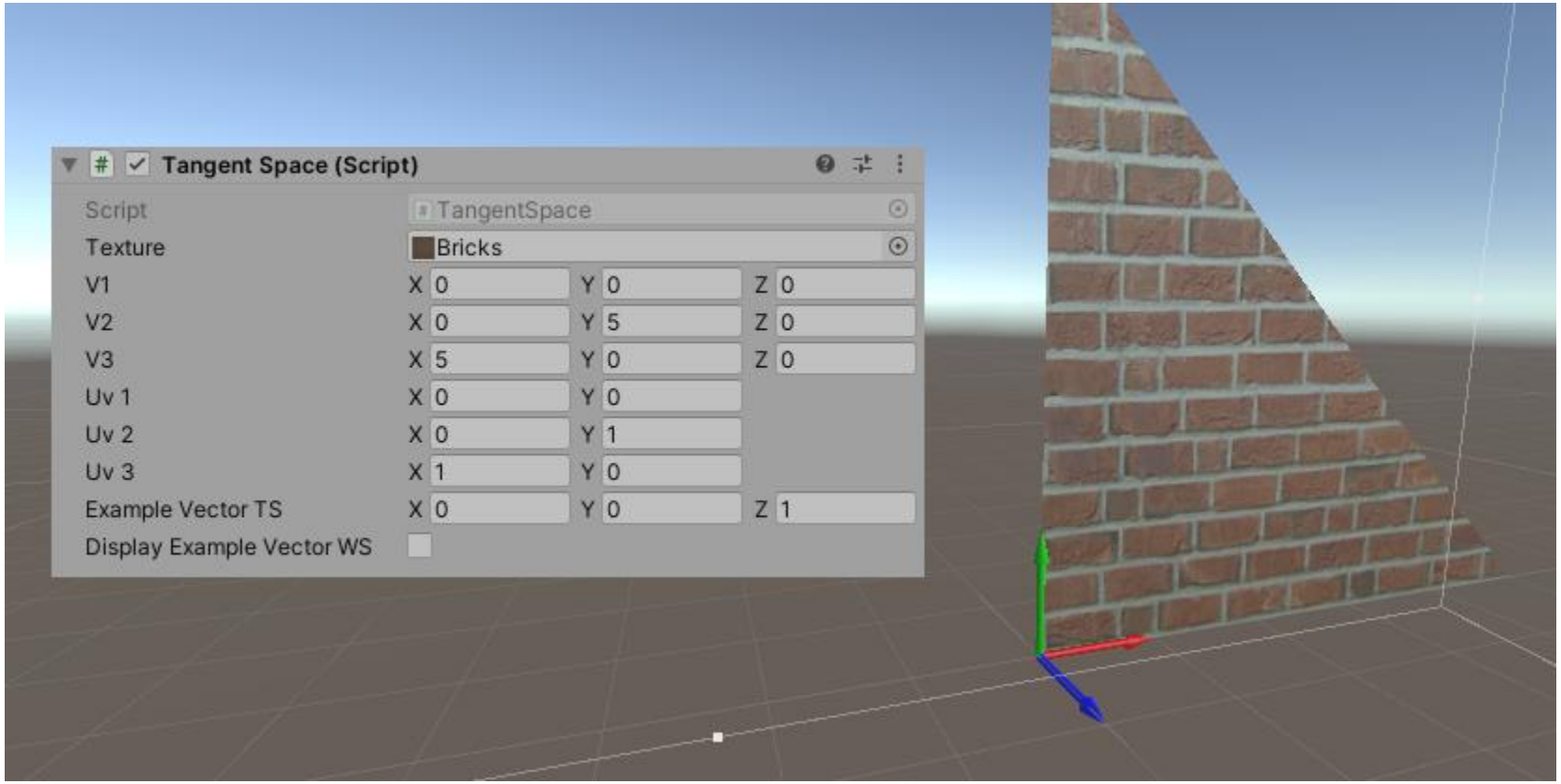
```
private Vector4 GetPlaneEquation(Vector3 v1, Vector3 v2, Vector3 v3)
{
    Vector3 normal = Vector3.Cross(v2 - v1, v3 - v1).normalized;

    Vector4 planeEq;
    planeEq.x = normal.x;
    planeEq.y = normal.y;
    planeEq.z = normal.z;
    planeEq.w = -(planeEq.x * v1.x + planeEq.y * v1.y + planeEq.z * v1.z);

    return planeEq;
}
```

Copyright © by Wojtek Sterna

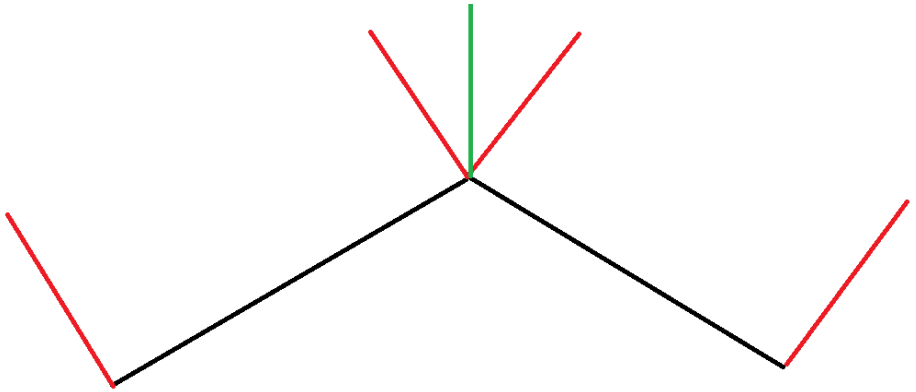
8.10. Tangent Space

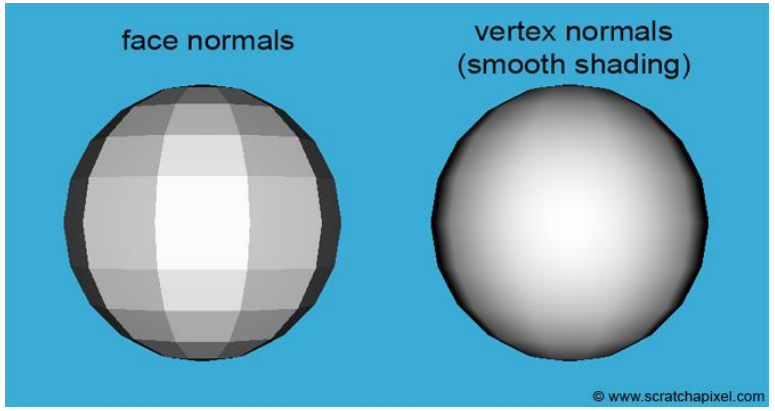


Copyright © by Wojtek Sterna

8.10. Tangent Space

- We've just discussed two different algorithms for generating the tangent space for a single triangle
- When generating tangent space for an entire 3D model we have to take into account averaging at the vertices:





- https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/shading-normals.html

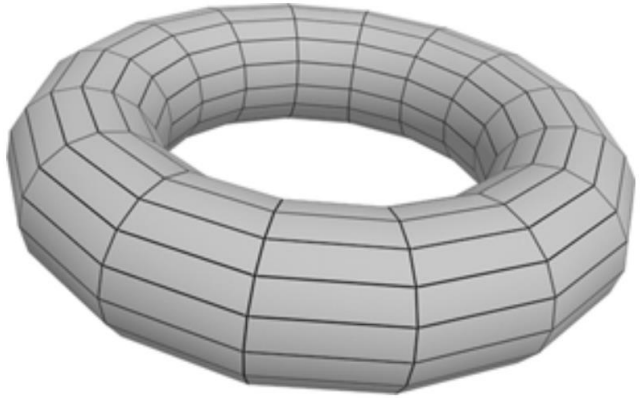
Copyright © by Wojtek Sterna

8.10. Tangent Space

- When calculating partial derivatives for the parametric plane equation we had:

$$\frac{\partial x}{\partial u} = \vec{t}_x \qquad \frac{\partial x}{\partial v} = \vec{b}_x \qquad \text{etc.}$$

- The above is actually true for any 3D surface that is represented with a parametric equation. Here we have torus (M is the outer radius, N is the inner radius):



Equation 8-3 Parametric Equations for a Torus

$$x = (M + N\cos(2\pi t))\cos(2\pi s)$$
$$y = (M + N\cos(2\pi t))\sin(2\pi s)$$
$$z = N\sin(2\pi t)$$

Equation 8-4 Partial Derivatives of the Parametric Torus

$$\frac{\partial x}{\partial s} = -2\pi(M + N\cos(2\pi t))\sin(2\pi s) \qquad \frac{\partial x}{\partial t} = -2N\pi\sin(2\pi t)\cos(2\pi s)$$
$$\frac{\partial y}{\partial s} = 2\pi(M + N\cos(2\pi t))\cos(2\pi s) \qquad \frac{\partial y}{\partial t} = -2N\pi\cos(2\pi t)\sin(2\pi s)$$
$$\frac{\partial z}{\partial s} = 0 \qquad \frac{\partial z}{\partial t} = 2N\pi\cos(2\pi t)$$

https://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter08.html

Copyright © by Wojtek Sterna

8.10. Tangent Space

- Tangent space in the whole transformations pipeline:

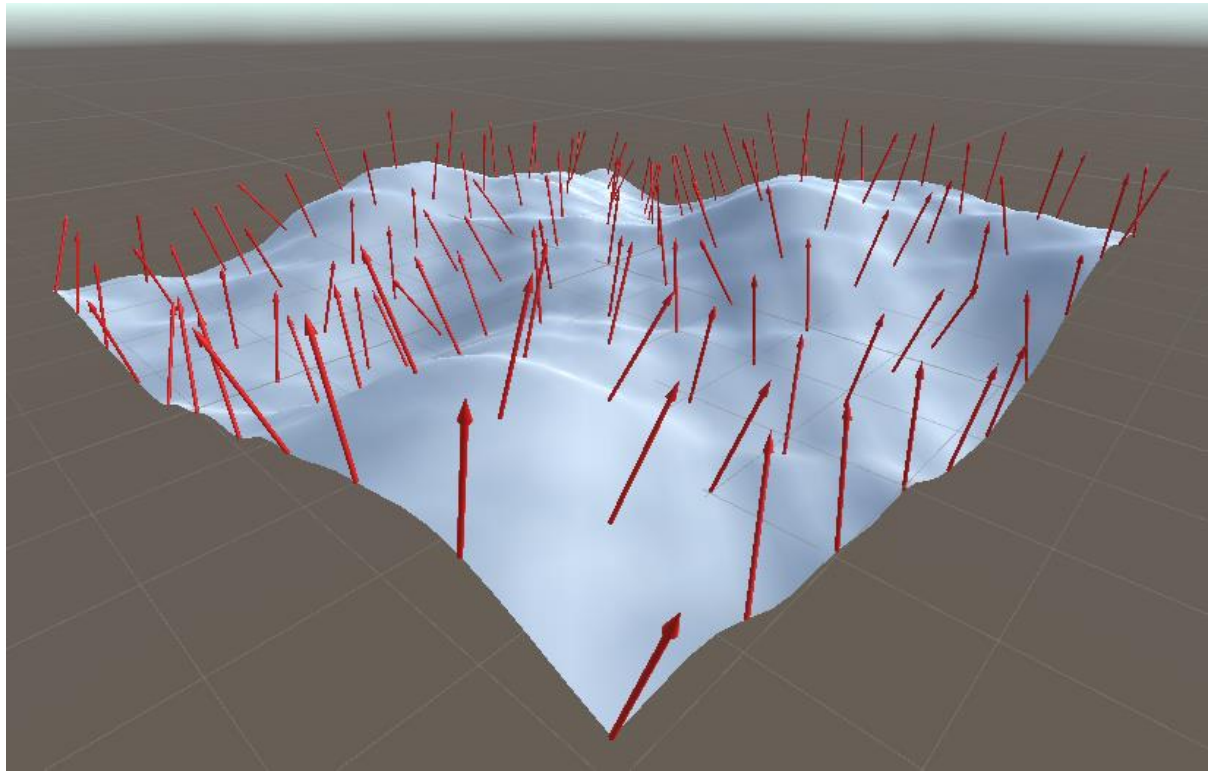
  tangent → local/object → world → view/camera → projection/clip → NDC

- https://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter08.html

Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- We will now write a program which generates a water surface – it will calculate vertices that form a 3D model as well as their normal vectors
- This program is a „3D version” of what we learned when we discussed the DerivativeParametric program and also a few moments ago



Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- The water surface is represented with the following parametric functions:

$$\begin{cases} x(u,v,t) = u \\ y(u,v,t) = \sum_{i=1}^{16} A_i \sin(B_i * (X_i * u + Y_i * v) + C_i * t + D_i) \\ z(u,v,t) = v \end{cases}$$

- Constants A_i, B_i, C_i, D_i are:
  the amplitude, spatial frequency, temporal frequency and phase offset of a single wave.
  Vector (X_i, Y_i) is the wave's move direction
- u and v are the functions' parameters, together with time t

Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- Code that calculates the constants values:

```
A = Zenon.Random(0.31f, 0.42f) * Mathf.Pow(0.73f, idx);
B = Zenon.Random(3.7f, 5.98f) * (1.0f + idx);
C = Zenon.Random(-0.5f, 0.5f);
D = Zenon.Random(-3.2f, 1.65f);
X = Zenon.Random(-0.1f, 0.1f);
Y = Zenon.Random(-0.1f, 0.1f);
```

- **We do not** focus here on good quality water surface generation itself but on the usage of derivatives in the tangent and normal vectors calculations
- More information about water generation itself:
  Turning Sine Waves Into Game Water
  https://developer.nvidia.com/gpugems/gpugems/part-i-natural-effects/chapter-1-effective-water-simulation-physical-models

Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- Vertex **position** calculation in code:

```
private Vector3 GetVertexPosition(float u, float v, float t)
{
    float x = u;
    float z = v;

    float y = 0.0f;
    for (int i = 0; i < waves.Length; i++)
    {
        y += waves[i].Eval(u, v, t);
    }

    return new Vector3(x, y, z);
}
```

```
public float Eval(float u, float v, float t)
{
    return A * Mathf.Sin(B*(X*u + Y*v) + C*t + D);
}
```

Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- The tangent and normal vectors are calculated as:

$$\vec{t} = \left[\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}\right] \qquad \vec{b} = \left[\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v}\right]$$

$$\vec{t} = [x_u, y_u, z_u] \qquad \vec{b} = [x_v, y_v, z_v]$$

$$\vec{n} = \vec{t} \times \vec{b}$$

Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- Derivative with respect to u:

$$\begin{cases} x_u(u,v,t) = 1 \\ y_u(u,v,t) = \sum_{i=1}^{16} A_i \cos(B_i * (X_i * u + Y_i * v) + C_i * t + D_i) * (B_i * X_i) \\ z_u(u,v,t) = 0 \end{cases}$$

Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- Derivative with respect to v:

$$\begin{cases} x_v(u,v,t) = 0 \\ y_v(u,v,t) = \sum_{i=1}^{16} A_i \cos(B_i * (X_i * u + Y_i * v) + C_i * t + D_i) * (B_i * Y_i) \\ z_v(u,v,t) = 1 \end{cases}$$

Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- Vertex **tangents** and **normal** calculation in code:

```
private Vector3 GetVertexNormal(float u, float v, float t)
{
    float dY_dU = 0.0f;
    float dY_dV = 0.0f;
    for (int i = 0; i < waves.Length; i++)
    {
        dY_dU += waves[i].EvalDerivativeU(u, v, t, derivativeMode);
        dY_dV += waves[i].EvalDerivativeV(u, v, t, derivativeMode);
    }

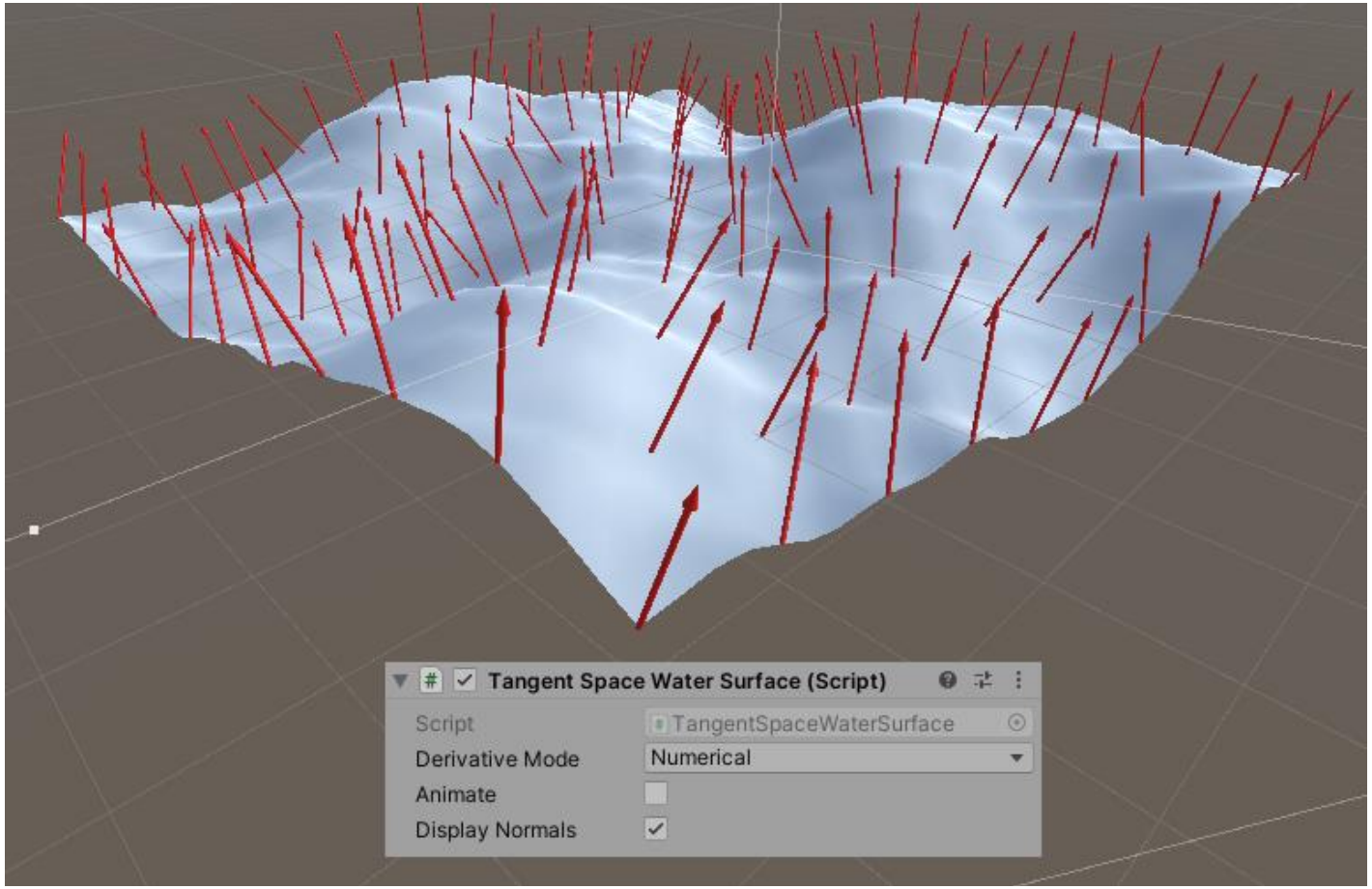
    Vector3 tangent = new Vector3(1.0f, dY_dU, 0.0f);
    Vector3 bitangent = new Vector3(0.0f, dY_dV, 1.0f);

    // negate, because (1, 0, 0) x (0, 0, 1) = (0, -1, 0)
    return -Vector3.Cross(tangent, bitangent).normalized;
}
```

```
public float EvalDerivativeU(float u, float v, float t, DerivativeMode derivativeMode)
{
    if (derivativeMode == DerivativeMode.Numerical)
    {
        return (Eval(u + 0.001f, v, t) - Eval(u - 0.001f, v, t)) / 0.002f;
    }
    else // Analytical
    {
        return A * Mathf.Cos(B*(X*u + Y*v) + C*t + D) * (B * X);
    }
}
```

Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface



Copyright © by Wojtek Sterna

8.10. Tangent Space – Water Surface

- The water surface is spanned/defined across the u and v parameters.
  That is why the partial derivatives of x, y and z functions, with respect to these parameters, give us the tangent vectors
- The third parameter t on the other hand has no such meaning. It does not „follow the surface”
- Parameter t is used only to deform/animate the surface and its derivative will tell us the velocity of that deformation/animation

Copyright © by Wojtek Sterna

8.11. Exercises

1. Write a program that finds a function's extrema.
   Results can be very approximate (slide 31)
2. In chapter 4 we discussed program PlaneMovement.
   Modify it so that it also displays a vector that points in the direction of the highest slope increase
3. In program Gradient implement numerical differentiation in place of analytical (slide 68)
4. In Program GradientDescentCircle, in the AB variant/mode, implement mini batching (slide 104)
5. In program GradientDescentParabolaPointDistance change the method of calculating the derivative from analytical to numerical (slide 107)
6. Write a program that generates and renders a torus (vertices are enough; no triangles needed), as well as calculates and displays normal vectors of the vertices (slide 140)

Copyright © by Wojtek Sterna

8.11. Exercises

7. Program TangentSpaceWaterSurface has the ability to display normal vectors.
   Add displaying of tangent vectors (slide 150)
8. In program TangentSpaceWaterSurface calculate the derivatives with respect to t and display the result (slide 150)
9. Write a program which renders a „terrain”. Data about vertices heights should come from a 2D texture or an ordinary 2D array.
   Additionally, display the normal vector at each vertex of the terrain