Math for Game Programmers: Dual Numbers

Gino van den Bergen
gino@dtecta.com
Introduction

- Dual numbers extend real numbers, similar to complex numbers.
- Complex numbers adjoin an element $i$, for which $i^2 = -1$.
- Dual numbers adjoin an element $\varepsilon$, for which $\varepsilon^2 = 0$. 
Complex Numbers

- Complex numbers have the form
  \[ z = a + b \, i \]

  where \( a \) and \( b \) are real numbers.
- \( a = \text{real}(z) \) is the real part, and
- \( b = \text{imag}(z) \) is the imaginary part.
Complex Numbers (cont’d)

- Complex operations pretty much follow rules for real operators:
  - Addition:
    $$(a + b \, i) + (c + d \, i) = (a + c) + (b + d) \, i$$
  - Subtraction:
    $$(a + b \, i) - (c + d \, i) = (a - c) + (b - d) \, i$$
Complex Numbers (cont’d)

- Multiplication:

\[(a + b \, i) \, (c + d \, i) = (ac - bd) + (ad + bc) \, i\]

- Products of imaginary parts feed back into real parts.
Dual Numbers

- Dual numbers have the form

\[ z = a + b \varepsilon \]

similar to complex numbers.

- \( a = \text{real}(z) \) is the real part, and
- \( b = \text{dual}(z) \) is the dual part.
Dual Numbers (cont’d)

- Operations are similar to complex numbers, however since $\varepsilon^2 = 0$, we have:

$$(a + b \varepsilon) (c + d \varepsilon) = (ac + 0) + (ad + bc)\varepsilon$$

- Dual parts do not feed back into real parts!
Dual Numbers (cont’d)

- The real part of a dual calculation is independent of the dual parts of the inputs.

- The dual part of a multiplication is a “cross” product of real and dual parts.
Taylor Series

- Any value $f(a + h)$ of a smooth function $f$ can be expressed as an infinite sum:

$$f(a + h) = f(a) + \frac{f'(a)}{1!} h + \frac{f''(a)}{2!} h^2 + \cdots$$

where $f'$, $f''$, ..., $f^{(n)}$ are the first, second, ..., $n$-th derivative of $f$. 
Taylor Series Example
Taylor Series Example
Taylor Series Example
Taylor Series Example
Taylor Series Example
Taylor Series and Dual Numbers

- For $f(a + b \varepsilon)$, the Taylor series is:

$$f(a + b\varepsilon) = f(a) + \frac{f'(a)}{1!} b\varepsilon + \ldots 0$$

- All second- and higher-order terms vanish!

- We have a closed-form expression that holds the function and its derivative.
Real Functions on Dual Numbers

- Any differentiable real function $f$ can be extended to dual numbers, as:

$$f(a + b \varepsilon) = f(a) + b f'(a) \varepsilon$$

- For example,

$$\sin(a + b \varepsilon) = \sin(a) + b \cos(a) \varepsilon$$
Automatic Differentiation

- Add a unit dual part to the input value of a real function.
- Evaluate function using dual arithmetic.
- The output has the function value as real part and the derivate’s value as dual part:

\[ f(a + \varepsilon) = f(a) + f'(a) \varepsilon \]
How does it work?

- Check out the product rule of differentiation: 
  \[(f \cdot g)' = f \cdot g' + f' \cdot g\]
- Notice the “cross” product of functions and their derivatives.
- Recall that
  \[(a + a'\varepsilon)(b + b'\varepsilon) = ab + (ab' + a'b)\varepsilon\]
Automatic Differentiation in C++

- We need some easy way of extending functions on floating-point types to dual numbers...
- ...and we need a type that holds dual numbers and offers operators for performing dual arithmetic.
Extension by Abstraction

- C++ allows you to abstract from the numerical type through:
  - Typedefs
  - Function templates
  - Constructors and conversion operators
  - Overloading
  - Traits class templates
Abstract Scalar Type

- Never use built-in floating-point types, such as `float` or `double`, explicitly.
- Instead use a type name, e.g. `Scalar`, either as template parameter or as `typedef`:

```cpp
typedef float float Scalar;
```
Constructors

- Built-in types have constructors as well:
  - Default: `float() == 0.0f`
  - Conversion: `float(2) == 2.0f`

- Use constructors for defining constants, e.g. use `Scalar(2)`, rather than `2.0f` or `(Scalar)2`.
Overloading

- Operators and functions on built-in types can be overloaded in numerical classes, such as `std::complex`.
- Built-in types support operators: `+,-,*,/`
- ...and functions: `sqrt`, `pow`, `sin`, ...
- NB: Use `<cmath>` rather than `<math.h>`. That is, use `sqrt` NOT `sqrtf` on floats.
Traits Class Templates

- Type-dependent constants, such as the machine epsilon, are obtained through a traits class defined in `<limits>`.
- Use `std::numeric_limits<Scalar>::epsilon()` rather than `FLT_EPSILON` in C++.
- Either specialize `std::numeric_limits` for your numerical classes or write your own traits class.
Example Code (before)

```c
float smoothstep(float x)
{
    if (x < 0.0f)
        x = 0.0f;
    else if (x > 1.0f)
        x = 1.0f;
    return (3.0f - 2.0f * x) * x * x;
}
```
Example Code (after)

template <typename T>
T smoothstep(T x)
{
    if (x < T())
        x = T();
    else if (x > T(1))
        x = T(1);
    return (T(3) - T(2) * x) * x * x * x;
}
Dual Numbers in C++

- C++ has a standard class template `std::complex<T>` for complex numbers.
- We create a similar class template `Dual<T>` for dual numbers.
- `Dual<T>` defines constructors, accessors, operators, and standard math functions.
Dual<T>

template <typename T>
class Dual
{
    ...

    private:
        T mReal;
        T mDual;

};
Dual<T>:: Constructor

template<typename T>
Dual<T>::::Dual(T real = T(), T dual = T())
    : mReal(real)
    , mDual(dual)
{}

...  
Dual<Scalar> z1; // zero initialized
Dual<Scalar> z2(2); // zero dual part
Dual<Scalar> z3(2, 1);
template <typename T>
Dual<T> operator*(Dual<T> a, Dual<T> b) {
    return Dual<T>(
        a.real() * b.real(),
        a.real() * b.dual() +
        a.dual() * b.real() );
}
Dual<T>: Standard Math

template<typename T>
Dual<T> sqrt(Dual<T> z)
{
    T tmp = sqrt(z.real());
    return Dual<T>(
        tmp,
        z.dual() * T(0.5) / tmp
    );
}
Curve Tangent

- For a 3D curve

\[ p(t) = (x(t), y(t), z(t)), \text{ where } t \in [a, b] \]

The tangent is

\[ \frac{p'(t)}{\|p'(t)\|}, \text{ where } p'(t) = (x'(t), y'(t), z'(t)) \]
Curve Tangent

- Curve tangents are often computed by approximation:

\[
\frac{p(t_1) - p(t_0)}{\|p(t_1) - p(t_0)\|}, \quad \text{where} \quad t_1 = t_0 + h
\]

for tiny values of h.
Curve Tangent: Bad #1

$P(t_0)$

$P(t_1)$

Actual tangent
Curve Tangent: Bad #2

$t_1$ drops outside parameter domain ($t_1 > b$)
Curve Tangent: Duals

- Make a curve function template using a class template for 3D vectors:

```cpp
template <typename T>
Vector3<T> curveFunc(T x);
```
Curve Tangent: Duals (cont’d)

- Call the curve function using a dual number \( x = \text{Dual}<\text{Scalar}>(t, 1) \), (add \( \varepsilon \) to parameter \( t \)):

\[
\text{Vector3}<\text{Dual}<\text{Scalar}> > y = \text{curveFunc}(<\text{Dual}<\text{Scalar}>(t, 1));
\]
Curve Tangent: Duals (cont’d)

- The real part is the evaluated position:
  \[
  \text{Vector3<Scalar> position} = \text{real}(y);
  \]

- The normalized dual part is the tangent at this position:
  \[
  \text{Vector3<Scalar> tangent} = \text{normalize}((\text{dual}(y)));
  \]
Line Geometry

- The line through points $\mathbf{p}$ and $\mathbf{q}$ can be expressed explicitly as:

$$x(t) = p + (q - p)t,$$

and

- Implicitly, as a set of points $\mathbf{x}$ for which:

$$(q - p) \times x + p \times q = 0$$
Line Geometry

\( \mathbf{p} \times \mathbf{q} \) is orthogonal to the plane \( \mathbf{opq} \), and its length is equal to the area of the parallelogram spanned by \( \mathbf{p} \) and \( \mathbf{q} \).
Line Geometry

All points $x$ on the line $pq$ span with $q - p$ a parallelogram that has the same area and orientation as the one spanned by $p$ and $q$. 
Plücker Coordinates

- Plücker coordinates are 6-tuples of the form $\mathbf{u} = (u_x, u_y, u_z, v_x, v_y, v_z)$, where

  $$\mathbf{u} = (u_x, u_y, u_z) = \mathbf{q} - \mathbf{p}, \quad \text{and}$$

  $$\mathbf{v} = (v_x, v_y, v_z) = \mathbf{p} \times \mathbf{q}$$
Plücker Coordinates (cont’d)

• For \((u_1:v_1)\) and \((u_2:v_2)\) directed lines, if

\[ u_1 \cdot v_2 + v_1 \cdot u_2 \]

is zero: the lines intersect

positive: the lines cross right-handed

negative: the lines cross left-handed
If the signs of permuted dot products of the ray and edges are all equal, then the ray intersects the triangle.
Plücker Coordinates and Duals

- Dual 3D vectors conveniently represent Plücker coordinates:

\[
\text{Vector3<\text{Dual<Scalar}> >}
\]

- For a line \((\mathbf{u}:\mathbf{v})\), \(\mathbf{u}\) is the real part and \(\mathbf{v}\) is the dual part.
Dot Product of Dual Vectors

- The dot product of dual vectors \( u_1 + v_1 \varepsilon \) and \( u_2 + v_2 \varepsilon \) is a dual number \( z \), for which

\[
\text{real}(z) = u_1 \cdot u_2, \quad \text{and} \quad \text{dual}(z) = u_1 \cdot v_2 + v_1 \cdot u_2
\]

- The dual part is the permuted dot product...
Angle of Dual Vectors

- For \( \mathbf{a} \) and \( \mathbf{b} \) dual vectors, we have

\[
\theta + d\varepsilon = \arccos\left( \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} \right)
\]

where \( \theta \) is the angle and \( d \) is the signed distance between the lines \( \mathbf{a} \) and \( \mathbf{b} \).
Translation

• Translation of lines only affects the dual part. Translation of line $pq$ over $c$ gives:

  • Real: $(q + c) - (p + c) = q - p$
  • Dual: $(p + c) \times (q + c) = p \times q + c \times (q - p)$

• $q - p$ pops up in the dual part!
Rotation

• Real and dual parts are rotated in the same way. For a rotation matrix $\mathbf{R}$:
  
  • Real: $\mathbf{R}q - \mathbf{R}p = \mathbf{R}(q - p)$
  
  • Dual: $\mathbf{R}p \times \mathbf{R}q = \mathbf{R}(p \times q)$

• The latter holds for rotations only! That is, $\mathbf{R}$ performs no scaling or reflection.
Rigid-Body Transform

• For rotation matrix $\mathbf{R}$ and translation vector $\mathbf{c}$, the dual $3 \times 3$ matrix $\mathbf{M}$ with $\text{real}(\mathbf{M}) = \mathbf{R}$, and

\[
\text{dual}(\mathbf{M}) = [\mathbf{c}] \times \mathbf{R} =
\begin{bmatrix}
0 & -c_z & c_y \\
c_z & 0 & -c_x \\
-c_y & c_x & 0
\end{bmatrix}
\]

maps Plücker coordinates to the new reference frame.
Screw Theory

- A screw motion is a rotation about a line and a translation along the same line.

- “Any rigid body displacement can be defined by a screw motion.” (Chasles)
Chasles’ Theorem (Sketchy Proof)

- Decompose translation into a term along the line and a term orthogonal to the line.
- Translation orthogonal to the axis of rotation offsets the axis.
- Translation along the axis does not care about the position of the axis.
Translations Orthogonal to Axis
Example: Rolling Ball
Dual Quaternions

- Unit dual quaternions represent screw motions.
- The rigid body transform over a unit quaternion $q$ and vector $t$ is:

$$q + \frac{1}{2} t q \varepsilon$$

Here, $t$ is a quaternion with zero scalar part.
Where is the Screw?

- A unit dual quaternion can be written as

\[
\cos\left(\frac{\theta + d\varepsilon}{2}\right) + \sin\left(\frac{\theta + d\varepsilon}{2}\right)(u + v\varepsilon)
\]

where \(\theta\) is the rotation angle, \(d\), the translation distance, and \(u + v\varepsilon\), the line given in Plücker coordinates.
Rigid-Body Transform Revisited

- Similar to 3D vectors, Plücker coordinates can be transformed using dual quaternions.
- The mapping of a dual vector $\mathbf{v}$ according to a screw motion $\mathbf{q}$ is

$$\mathbf{v}' = \mathbf{q} \mathbf{v} \mathbf{q}^*$$
Traditional Skinning

- Bones are defined by transformation matrices $T_i$ relative to the rest pose.
- Each vertex is transformed as

$$p' = \lambda_1 T_1 p + \cdots + \lambda_n T_n p = (\lambda_1 T_1 + \cdots + \lambda_n T_n)p$$

Here, $\lambda_i$ are blend weights.
Traditional Skinning (cont’d)

- A weighted sum of matrices is not necessarily a rigid-body transformation.
- Most notable artifact is “candy wrapper”: The skin collapses while transiting from one bone to the other.
Candy Wrapper
Dual Quaternion Skinning

- Use a blend operation that always returns a rigid-body transformation.
- Several options exist. The simplest one is a normalized lerp of dual quaternions:

\[ q = \frac{\lambda_1 q_1 + \cdots + \lambda_n q_n}{\|\lambda_1 q_1 + \cdots + \lambda_n q_n\|} \]
Dual Quaternion Skinning (cont’d)

- Can the weighted sum of dual quaternions ever get zero?
- Not if all dual quaternions lie in the same hemisphere.
- Observe that \( q \) and \(-q\) are the same pose. If necessary, negate each \( q_i \) to dot positively with \( q_0 \).
Further Uses

- **Motor Algebra**: Linear and angular velocity of a rigid body combined in a dual 3D vector.

- **Spatial Vector Algebra**: Featherstone uses 6D vectors for representing velocities and forces in robot dynamics.
Conclusions

• Abstract from numerical types in your C++ code.
• Differentiation is easy, fast, and exact with dual numbers.
• Dual numbers have other uses as well. Explore yourself!
References

Thank You!

- For sample code, check out free* MoTo C++ template library on:

  https://code.google.com/p/motion-toolkit/

(*) gratis (as in “free beer”) and libre (as in “free speech”)