Hubert Łabuda 236587  
Jan Maślak 236598

Testy klasy Quaternion

#[test]  
fn identity\_test() {  
 let quat = Quaternion::identity();  
 println!("\nQUATERNION IDENTITY TEST\n");  
 println!("Calling Quaternion::Identity() returns an identity quaternion: {}", quat.to\_string());  
 assert\_eq!(quat.real, 1.0);  
 assert\_eq!(quat.ivec.x, 0.0);  
 assert\_eq!(quat.ivec.y, 0.0);  
 assert\_eq!(quat.ivec.z, 0.0);  
}

Output:

QUATERNION IDENTITY TEST

Calling Quaternion::Identity() returns an identity quaternion: (1.00, 0.00i, 0.00j, 0.00k)

#[test]  
fn vector\_rotation\_test() {  
 let mut q = Quaternion::identity();  
 q.rotate(as\_radians(90.0), Vector::new(0.0, 0.0, 1.0));  
 let mut mat = Mat4::identity();  
 mat.rotate(as\_radians(90.0), Vector::new(0.0, 0.0, 1.0));  
 let mut v1 = Vector::new(1.0, 1.0, 1.0);  
 let mut v2 = Vector::new(1.0, 1.0, 1.0);  
 println!("\nQUATERNION VECTOR ROTATION TEST\n");  
 println!("Let's take a quaternion and a matrix rotated by 90deg on z axis: {},\n{}", q.to\_string(), mat.to\_string());  
 println!("Create two identical vectors: v1 = {},\nv2 = {}", v1.to\_string(), v2.to\_string());  
 println!("Rotate one of them with a quaternion and the second with a matrix");  
  
 q.rotate\_vec(&mut v1);  
 v2 = v2 \* mat;  
  
 println!("v1 = {}, v2 = {}", v1.to\_string(), v2.to\_string());  
 assert\_eq!(v1, v2);  
}

Output:

QUATERNION VECTOR ROTATION TEST

Let's take a quaternion and a matrix rotated by 90deg on z axis: (0.71, 0.00i, 0.00j, 0.71k),

[ 0.000 -1.000 0.000 0.000 ]

[ 1.000 0.000 0.000 0.000 ]

[ 0.000 0.000 1.000 0.000 ]

[ 0.000 0.000 0.000 1.000 ]

Create two identical vectors: v1 = [1.00, 1.00, 1.00],

v2 = [1.00, 1.00, 1.00]

Rotate one of them with a quaternion and the second with a matrix

v1 = [-1.00, 1.00, 1.00], v2 = [-1.00, 1.00, 1.00]

#[test]  
fn hamilton\_prod\_test() {  
 let mut q1 = Quaternion::new(1.0, Vector::new(2.0, 3.0, 4.0));  
 let mut q2 = Quaternion::new(2.0, Vector::new(3.0, 4.0, 5.0));  
  
 let h = q1.hamilton\_product(&q2);  
 let other\_h = q2.hamilton\_product(&q1);  
  
 println!("\nQUATERNION HAMILTON PRODUCT TEST\n");  
 println!("Let's take two quaternions: q1 = {}, q2 = {}", q1.to\_string(), q2.to\_string());  
  
 println!("Let's calculate their hamilton product.");  
 println!("the output of this operation should be:\n-36 + 6i + 12j + 12k if it's H(q1, q2)\nor -36 + 8i + 8j + 14k for H(q2, q1)");  
 println!("we revieved: H(q1, q2) = {}, H(q2, q1) = {}", h.to\_string(), other\_h.to\_string());  
  
 assert\_eq!(h, Quaternion::new(-36.0, Vector::new(6.0, 12.0, 12.0)));  
 assert\_eq!(other\_h, Quaternion::new(-36.0, Vector::new(8.0, 8.0, 14.0)));  
}

Output:

QUATERNION HAMILTON PRODUCT TEST

Let's take two quaternions: q1 = (1.00, 2.00i, 3.00j, 4.00k), q2 = (2.00, 3.00i, 4.00j, 5.00k)

Let's calculate their hamilton product.

the output of this operation should be:

-36 + 6i + 12j + 12k if it's H(q1, q2)

or -36 + 8i + 8j + 14k for H(q2, q1)

we revieved: H(q1, q2) = (-36.00, 6.00i, 12.00j, 12.00k), H(q2, q1) = (-36.00, 8.00i, 8.00j, 14.00k)

#[test]  
// Check if the rotations are correctly applied and then converted to rotation matrix  
fn to\_mat\_test() {  
 let mut quat = Quaternion::identity();  
 quat.rotate(as\_radians(90.0), Vector::new(1.0, 0.0, 0.0));  
 let quat\_mat = quat.to\_mat4();  
 println!("\nQUATERNION TO MATRIX TEST\n");  
 println!("Let's take a identity quaternion and rotate it by 90deg on x axis: {}", quat.to\_string());  
 println!("Then let's transform it into a mat4 frms: {}", quat\_mat.to\_string());  
  
  
 let mut other\_mat = Mat4::identity();  
 other\_mat.rotate(as\_radians(90.0), Vector::new(1.0, 0.0, 0.0));  
 println!("now let's take an identity matrix and perform the same operations {}", other\_mat.to\_string());  
 println!("they are the same, except for floating point precision errors");  
  
 let (mut vec1, mut vec2) = (Vector::new(1.0, 1.0, 1.0), Vector::new(1.0, 1.0, 1.0));  
 vec1 = vec1 \* quat\_mat;  
 vec2 = vec2 \* other\_mat;  
 assert\_eq!(vec1, vec2);  
}

Output

QUATERNION TO MATRIX TEST

Let's take a identity quaternion and rotate it by 90deg on x axis: (0.71, 0.71i, 0.00j, 0.00k)

Then let's transform it into a mat4 frms: [ 1.000 0.000 0.000 0.000 ]

[ 0.000 -0.000 -1.000 0.000 ]

[ 0.000 1.000 -0.000 0.000 ]

[ 0.000 0.000 0.000 0.000 ]

now let's take an identity matrix and perform the same operations [ 1.000 0.000 0.000 0.000 ]

[ 0.000 0.000 -1.000 0.000 ]

[ 0.000 1.000 0.000 0.000 ]

[ 0.000 0.000 0.000 1.000 ]

they are the same, except for floating point precision errors

#[test]  
fn normalize\_test() {  
 let mut q = Quaternion::new(1.0, Vector::new(2.0, 3.0, 4.0));  
 println!("\nQUATERNION NORMALIZE TEST\n");  
 println!("Let's take a quaternion: {}", q.to\_string());  
 println!("Let's normalize it");  
 q.normalize();  
 println!("now it's 'length' should be 1: {:.4}", q.length());  
 assert!(approx\_eq!(f64, q.length(), 1.0, F64Margin { epsilon: f64::EPSILON, ulps: 4 }));  
}

Output:  
QUATERNION NORMALIZE TEST

Let's take a quaternion: (1.00, 2.00i, 3.00j, 4.00k)

Let's normalize it

now it's 'length' should be 1: 1.0000

#[test]  
fn inverse\_test() {  
 let mut quat = Quaternion::new(1.0, Vector::new(2.0, 3.0, 4.0));  
 let mut inverse = quat.clone();  
 inverse.inverse();  
 println!("\nQUATERNION INVERSE TEST\n");  
 println!("Let's take a quaternion: {}", quat.to\_string());  
 println!("Let's calculate its inverse: {}", inverse.to\_string());  
 println!("Let's multiply the quaternion by its inverse");  
 quat \*= inverse;  
 println!("The result should be an identity quaternion: {}", quat.to\_string());  
 assert\_eq!(quat, Quaternion::new(1.0, Vector::new(0.0, 0.0, 0.0)));  
}

Output:

QUATERNION INVERSE TEST

Let's take a quaternion: (1.00, 2.00i, 3.00j, 4.00k)

Let's calculate its inverse: (0.03, -0.07i, -0.10j, -0.13k)

Let's multiply the quaternion by its inverse

The result should be an identity quaternion: (1.00, 0.00i, 0.00j, 0.00k)

#[test]  
fn conjugate\_test() {  
 let mut quat = Quaternion::new(1.0, Vector::new(2.0, 3.0, 4.0));  
 let mut conjugate = quat.clone();  
 conjugate.conjugate();  
 println!("\nQUATERNION CONJUGATE TEST\n");  
 println!("Let's take a quaternion: {}", quat.to\_string());  
 println!("Let's calculate its conjugate: {}", conjugate.to\_string());  
 assert\_eq!(conjugate, Quaternion::new(1.0, Vector::new(-2.0, -3.0, -4.0)));  
}

Output

QUATERNION CONJUGATE TEST

Let's take a quaternion: (1.00, 2.00i, 3.00j, 4.00k)

Let's calculate its conjugate: (1.00, -2.00i, -3.00j, -4.00k)