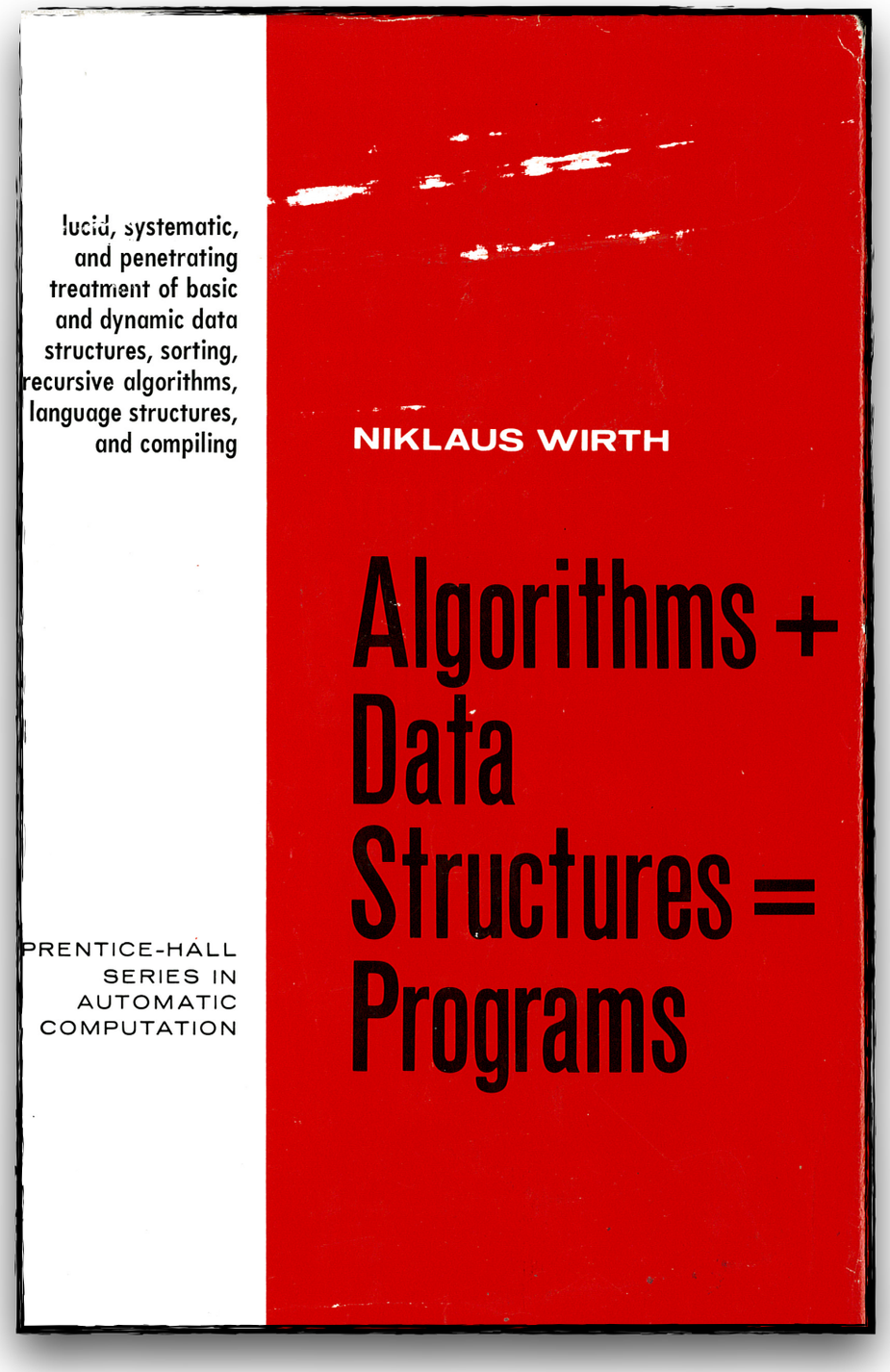
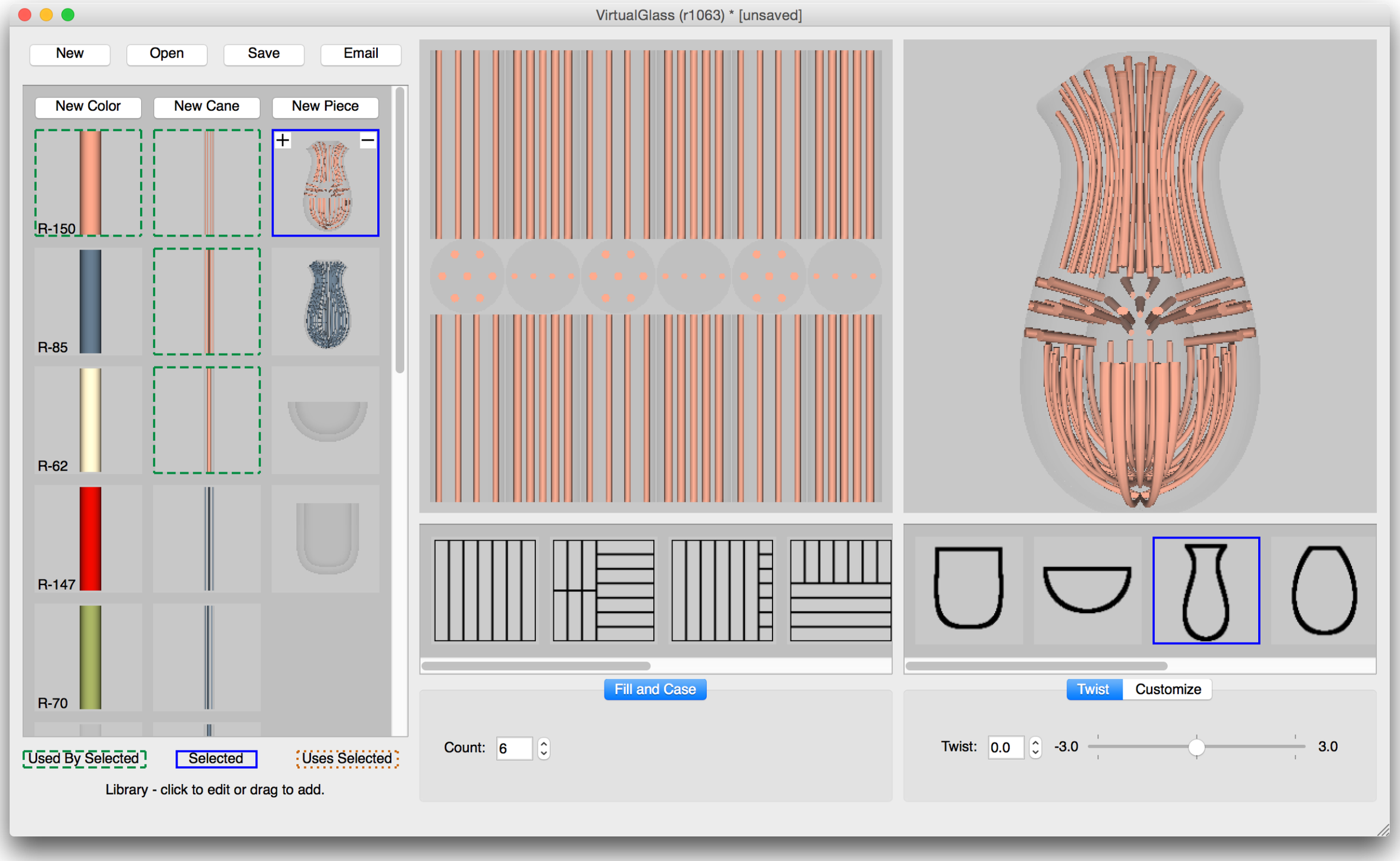
data & behavior



Is this a DSL?

VirtualGlass



The Expression Problem

There are multiple formulations. This is one of them.

- ∃ a library *L* of nouns (data) and verbs (behaviors)
- Person A extends *L* to add a new noun (data)
- Person B extends *L* to add a new verb (behavior)
- Person C wants to *safely* combine A and B's extensions

User-defined Types and Procedural Data Structures as Complementary Approaches to Data Abstraction, John Reynolds, 1975

Object-Oriented Programming Versus Abstract Data Types, William R. Cook, 1990

The Expression Problem, Philip Wadler, 1998

Independently Extensible Solutions to the Expression Problem, Matthias Zenger & Martin Odersky, 2005

Data types à la carte, Wouter Swierstra, 2008

CS 111 Spring ’16: The Playing Wildebeests

images.contentreserve.com/ImageType-400/0111-1/B1F/940/DA/%7BB1F940DA-E25D-4505-B2F8-CE106A55B455%7DImg400.jpg

DSL takeaways

1. We can use traits to mix in **syntax!**

ScalaTest = Traits for DSLs

Dialect: test-driven development

```
import org.scalatest.FunSuite
import scala.collection.mutable.Stack

class ExampleFunSuite extends FunSuite {

  test("pop is invoked on a non-empty stack") {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    val oldSize = stack.size
    val result = stack.pop()
    assert(result === 2)
    assert(stack.size === oldSize - 1)
  }

}
```

examples taken from scalatest.org

ScalaTest = Traits for DSLs

Dialect: behavior-driven development

```
import org.scalatest.FunSpec
import scala.collection.mutable.Stack

class ExampleFunSpec extends FunSpec {

  describe("A Stack") {

    it("should pop values in last-in-first-out order") {
      val stack = new Stack[Int]
      stack.push(1)
      stack.push(2)
      assert(stack.pop() === 2)
      assert(stack.pop() === 1)
    }

  }
}
```

examples taken from scalatest.org

ScalaTest = Traits for DSLs

Dialect: functional testing

```
import org.scalatest.FeatureSpec
import org.scalatest.GivenWhenThen
import scala.collection.mutable.Stack

class ExampleFeatureSpec extends FeatureSpec with GivenWhenThen {

  feature("The user can pop an element off the top of the stack") {

    info("As a programmer")
    info("I want to be able to pop items off the stack")
    info("So that I can get them in last-in-first-out order")

    scenario("pop is invoked on a non-empty stack") {

      given("a non-empty stack")
      val stack = new Stack[Int]
      stack.push(1)
      stack.push(2)
      val oldSize = stack.size

      when("when pop is invoked on the stack")
      val result = stack.pop()

      then("the most recently pushed element should be returned")
      assert(result === 2)

      and("the stack should have one less item than before")
      assert(stack.size === oldSize - 1)
    }
  }

}
```

examples taken from scalatest.org

DSL takeaways

1. We can use traits to mix in **syntax**!

2. We prefer case classes over classes.