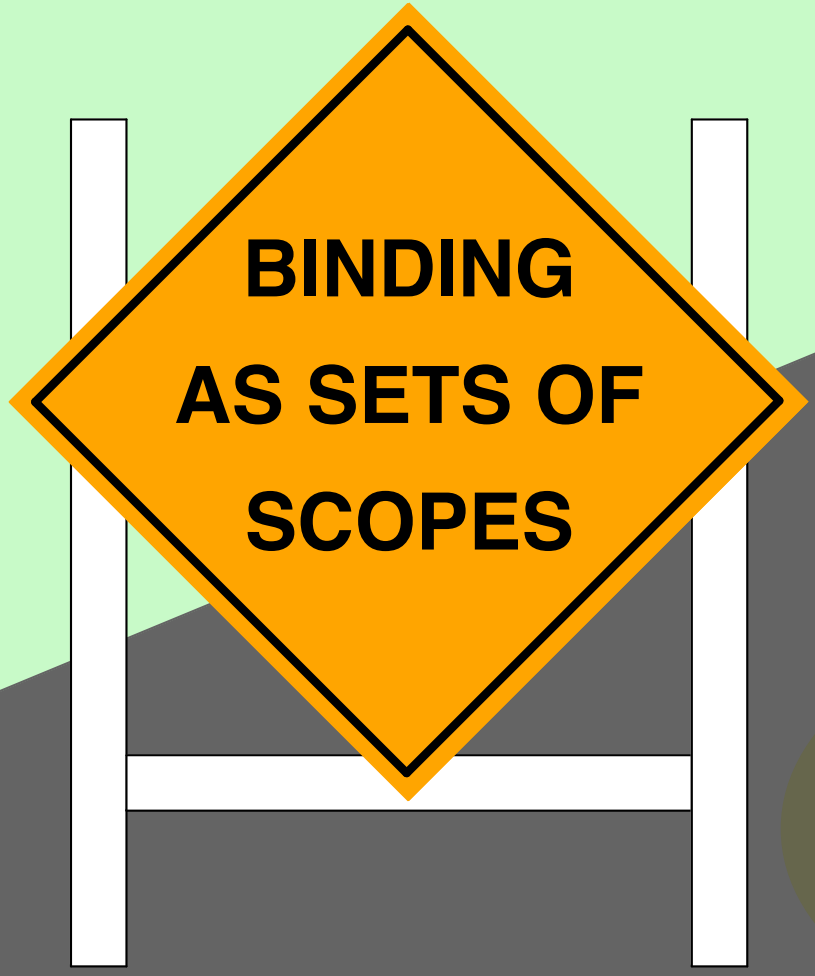
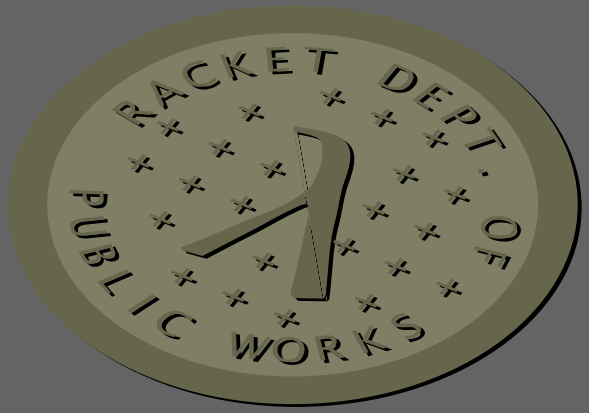
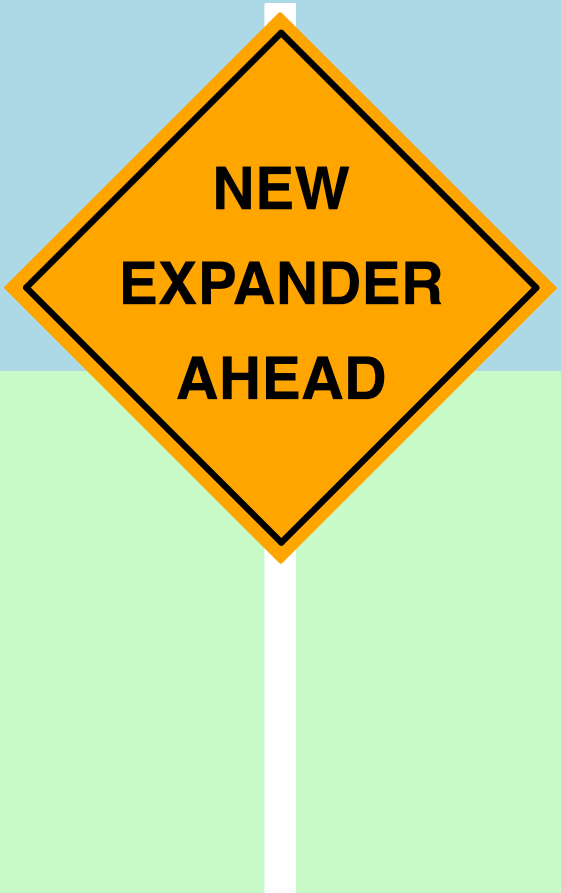


# BINDING AS SETS OF SCOPES




NEW
EXPANDER
AHEAD

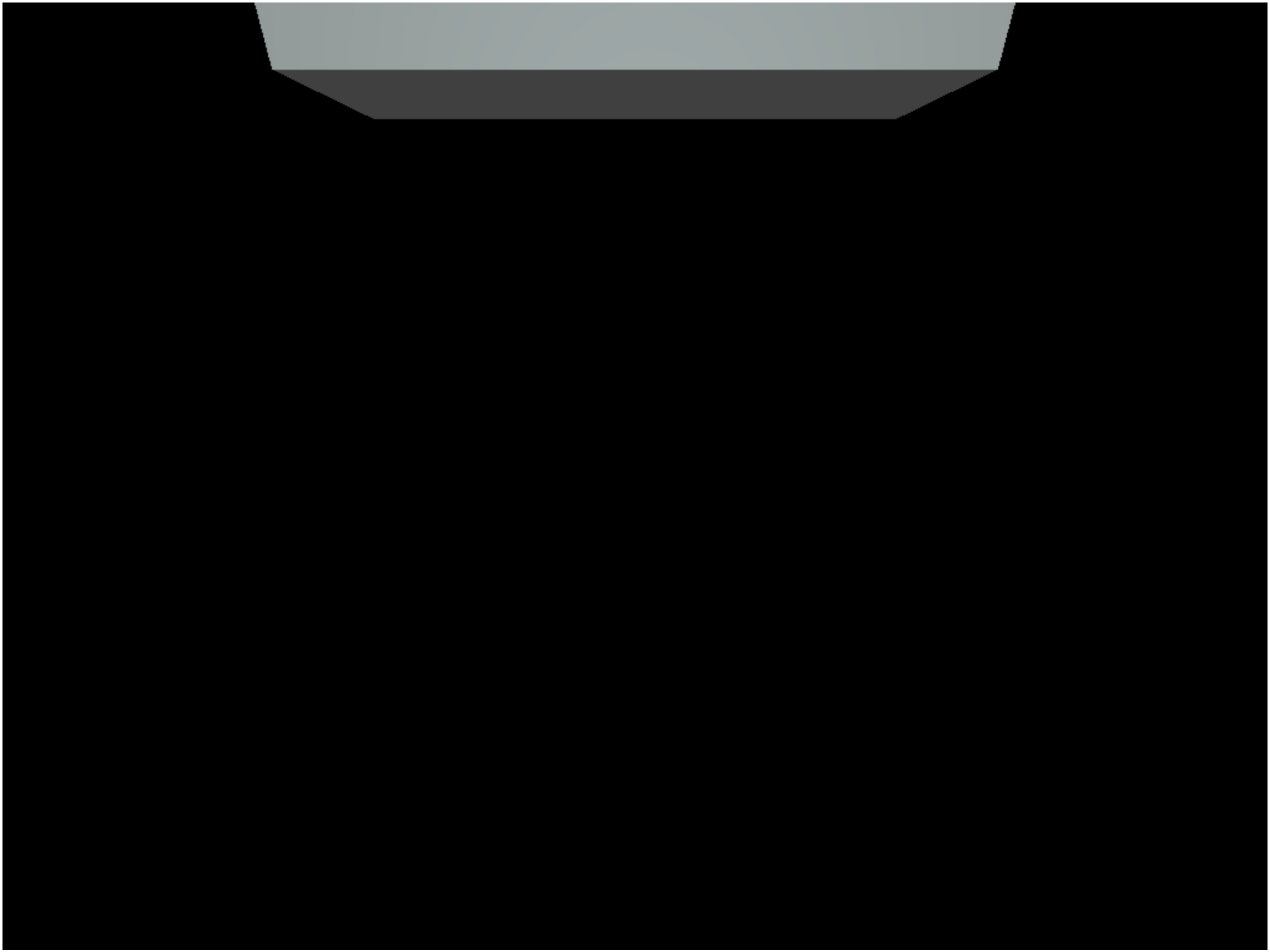
```
#lang typed/racket
```

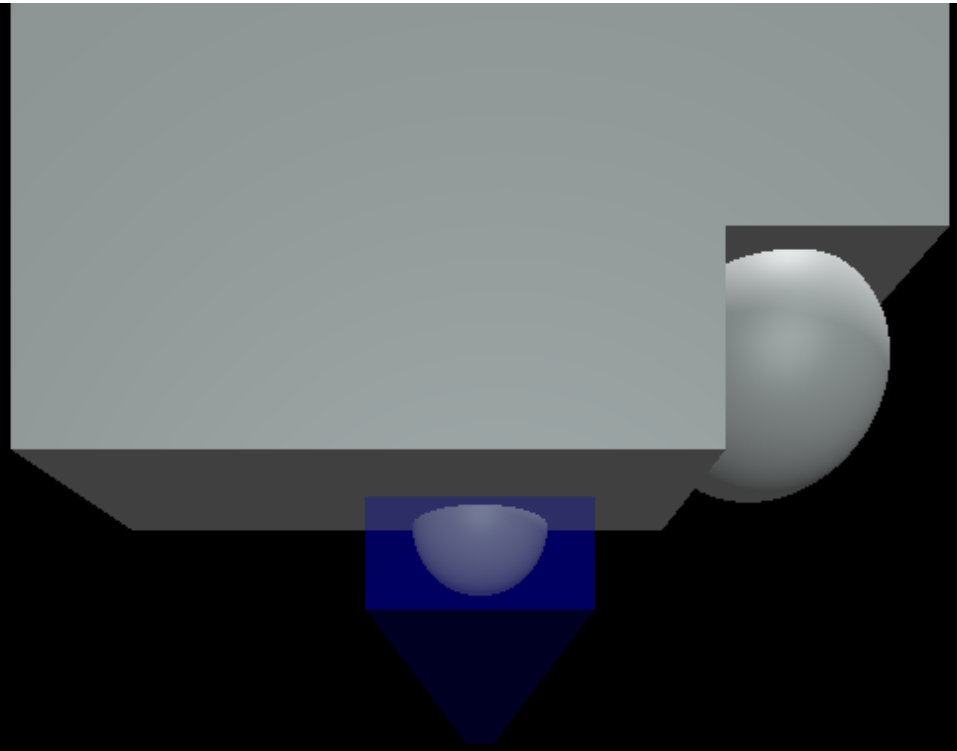
```
#lang typed/racket
```

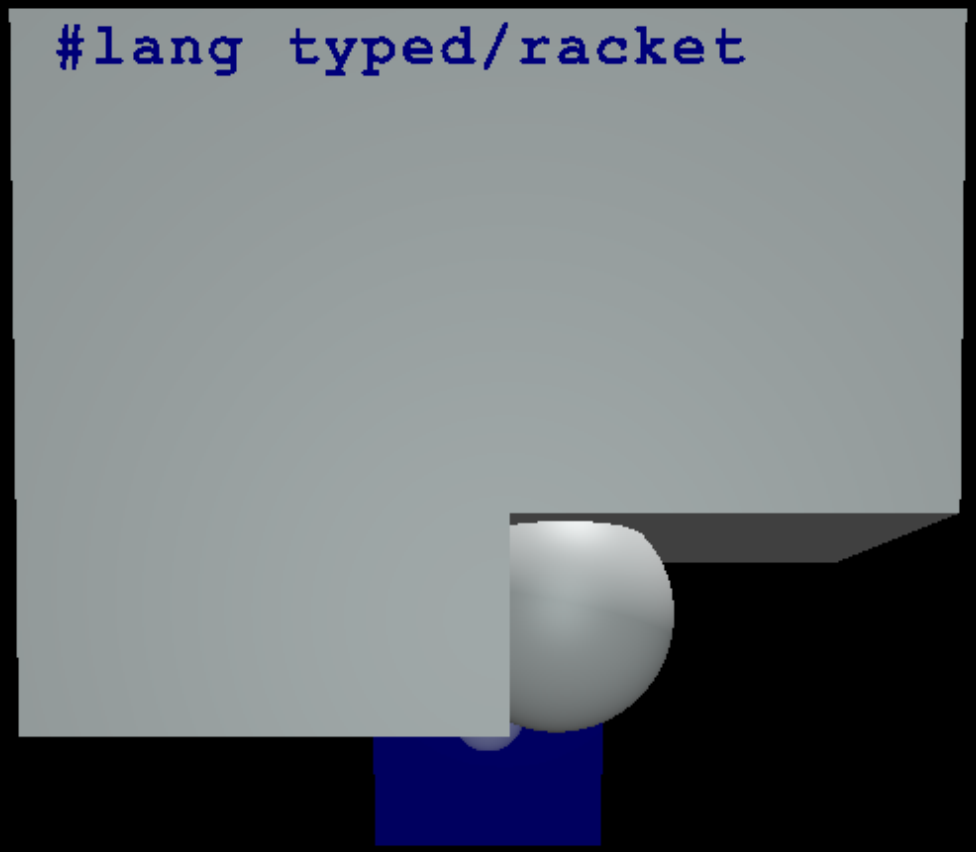
```
#lang typed/racket
```

```
#lang typed/racket
```

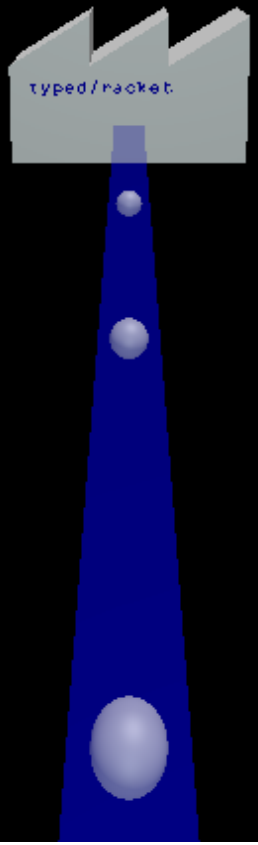
```
#lang typed/racket
```

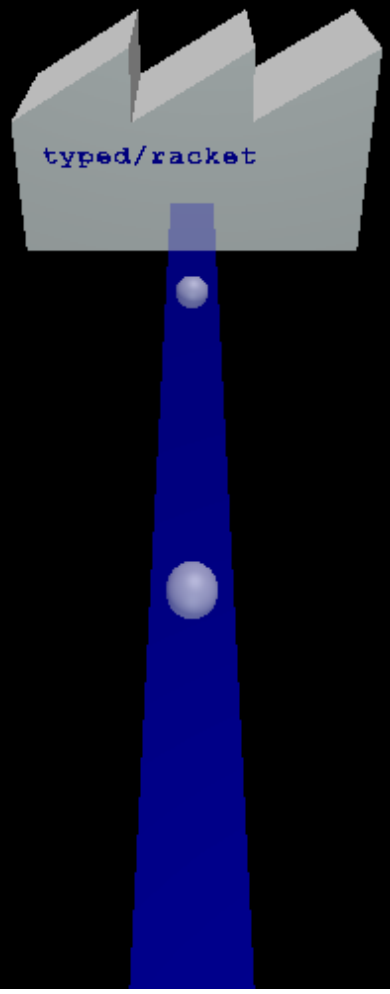


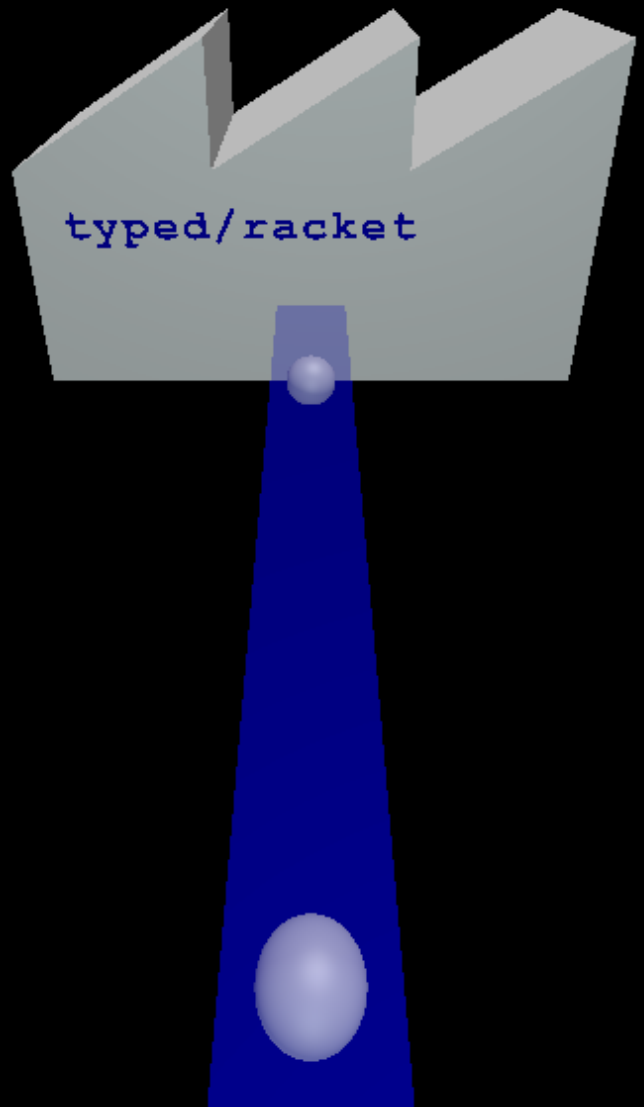


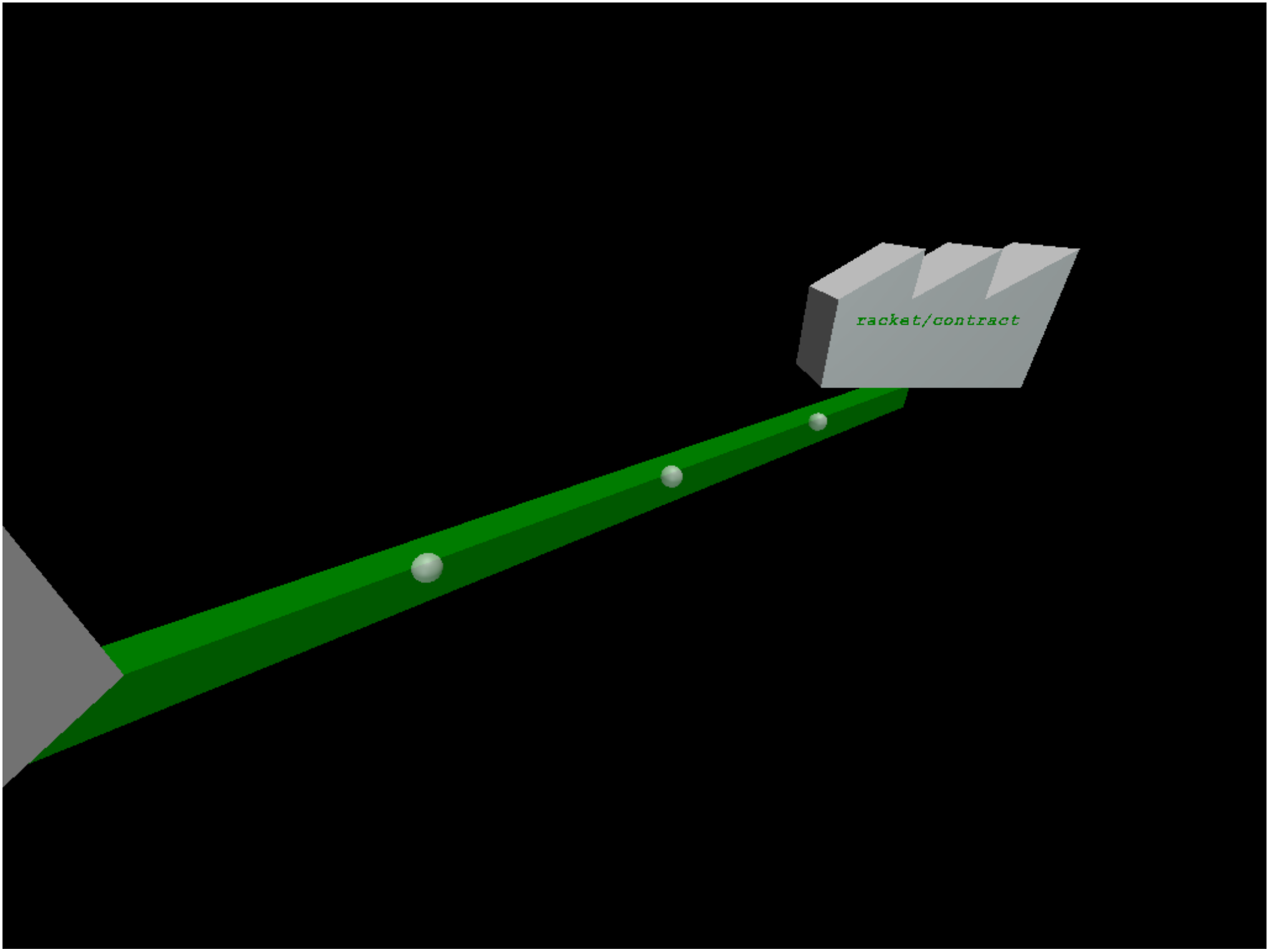

#lang typed/racket

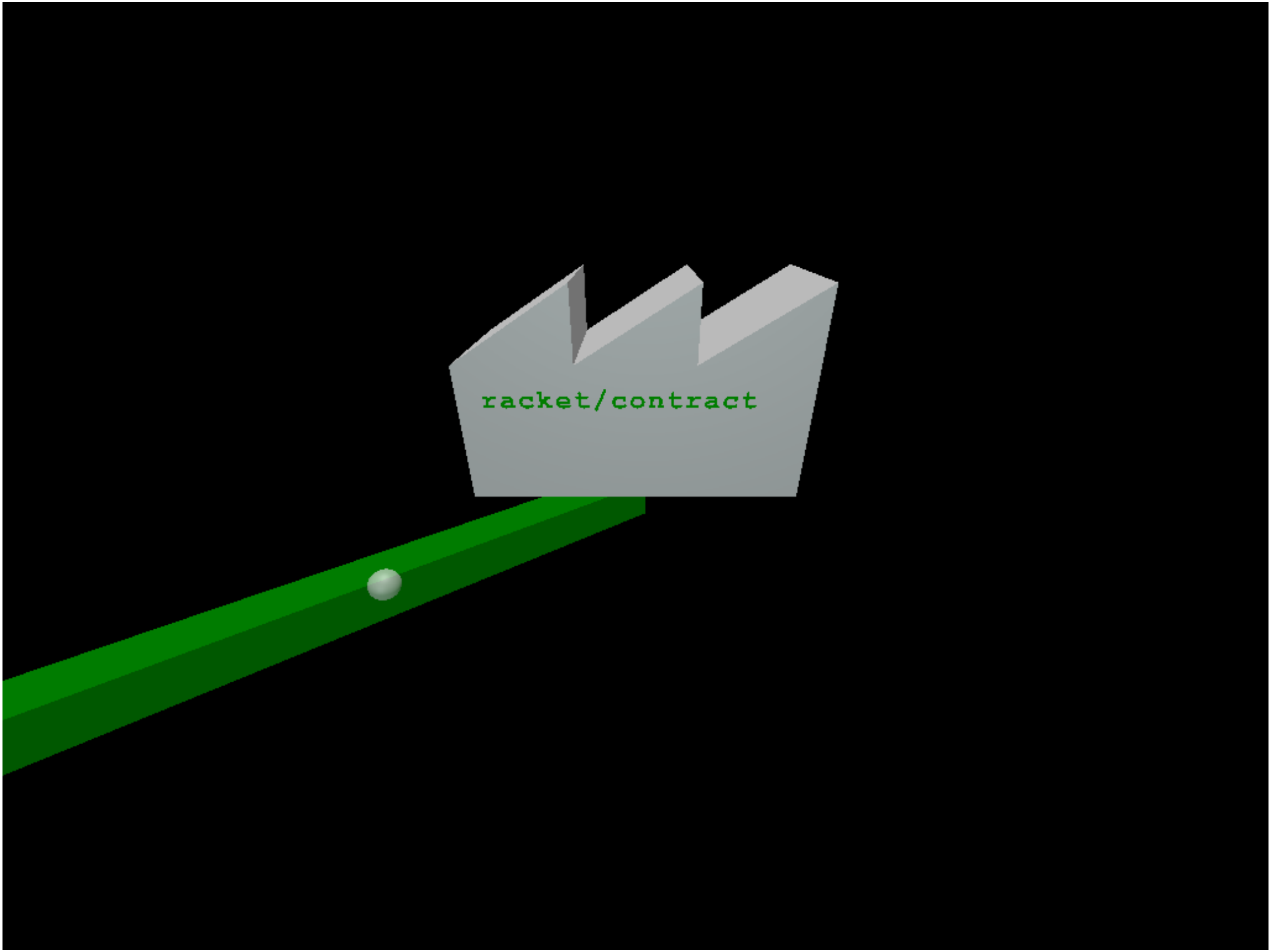
```
#lang typed/racket
```

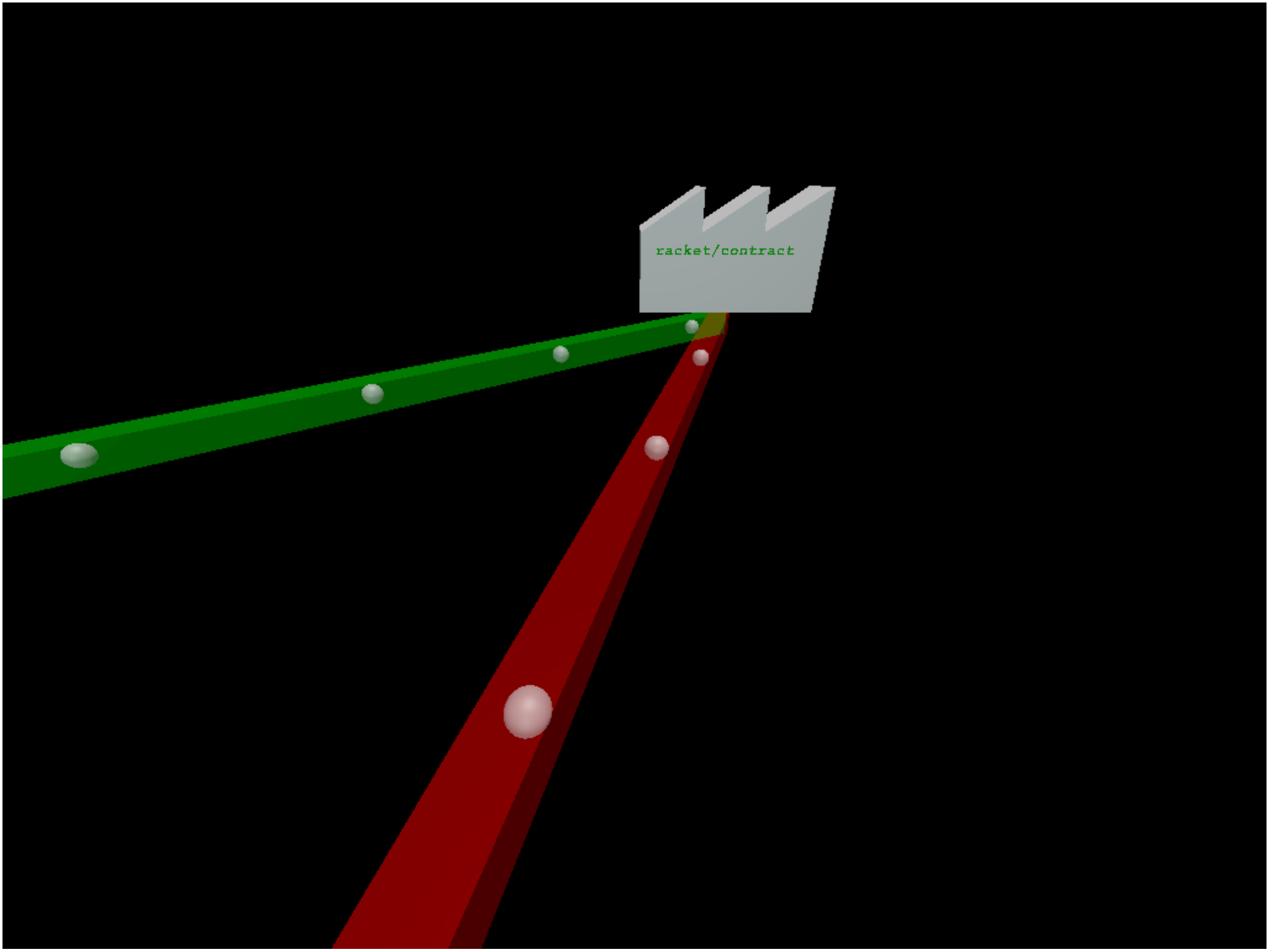

typed/racket

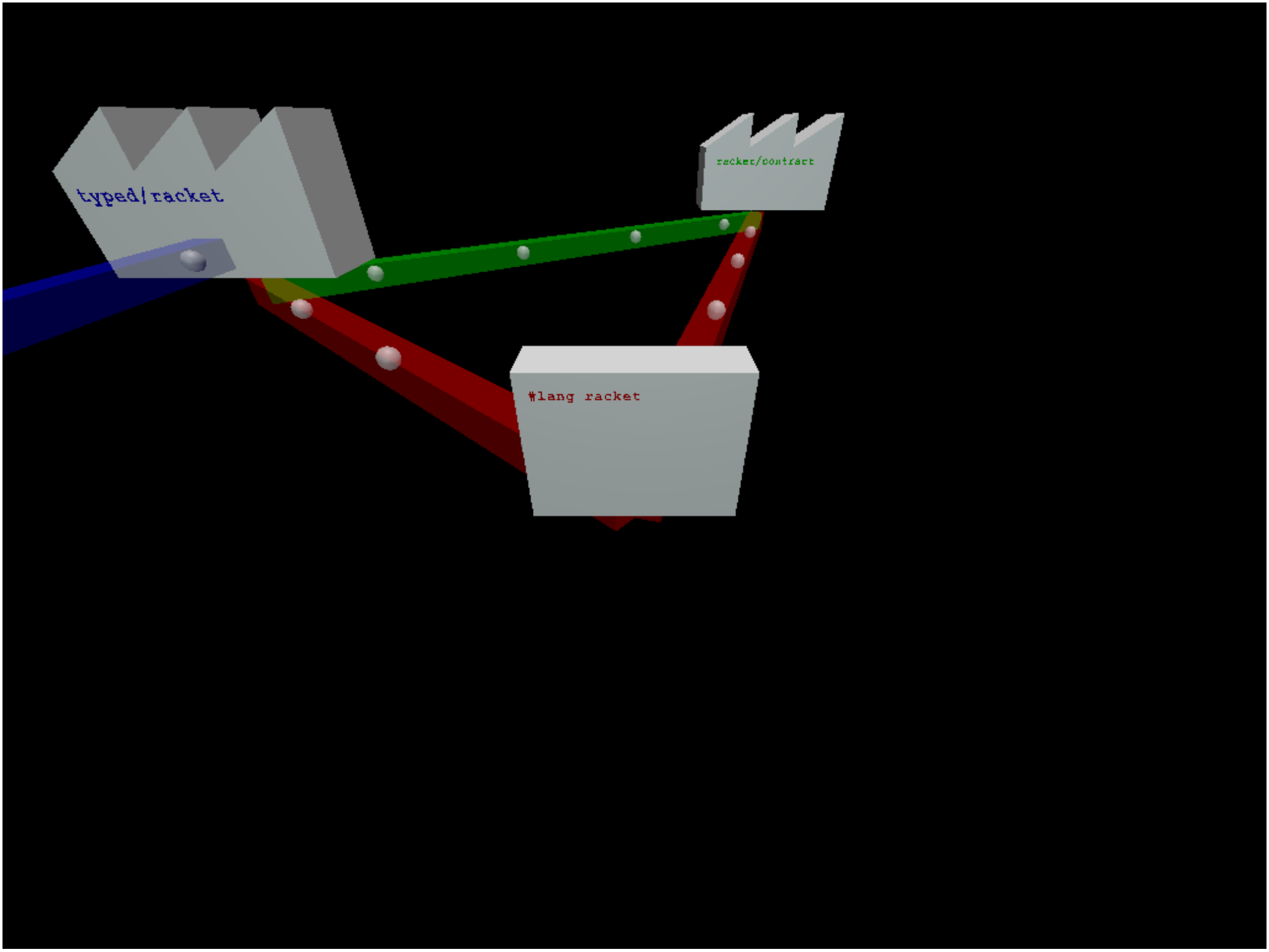

typed/racket

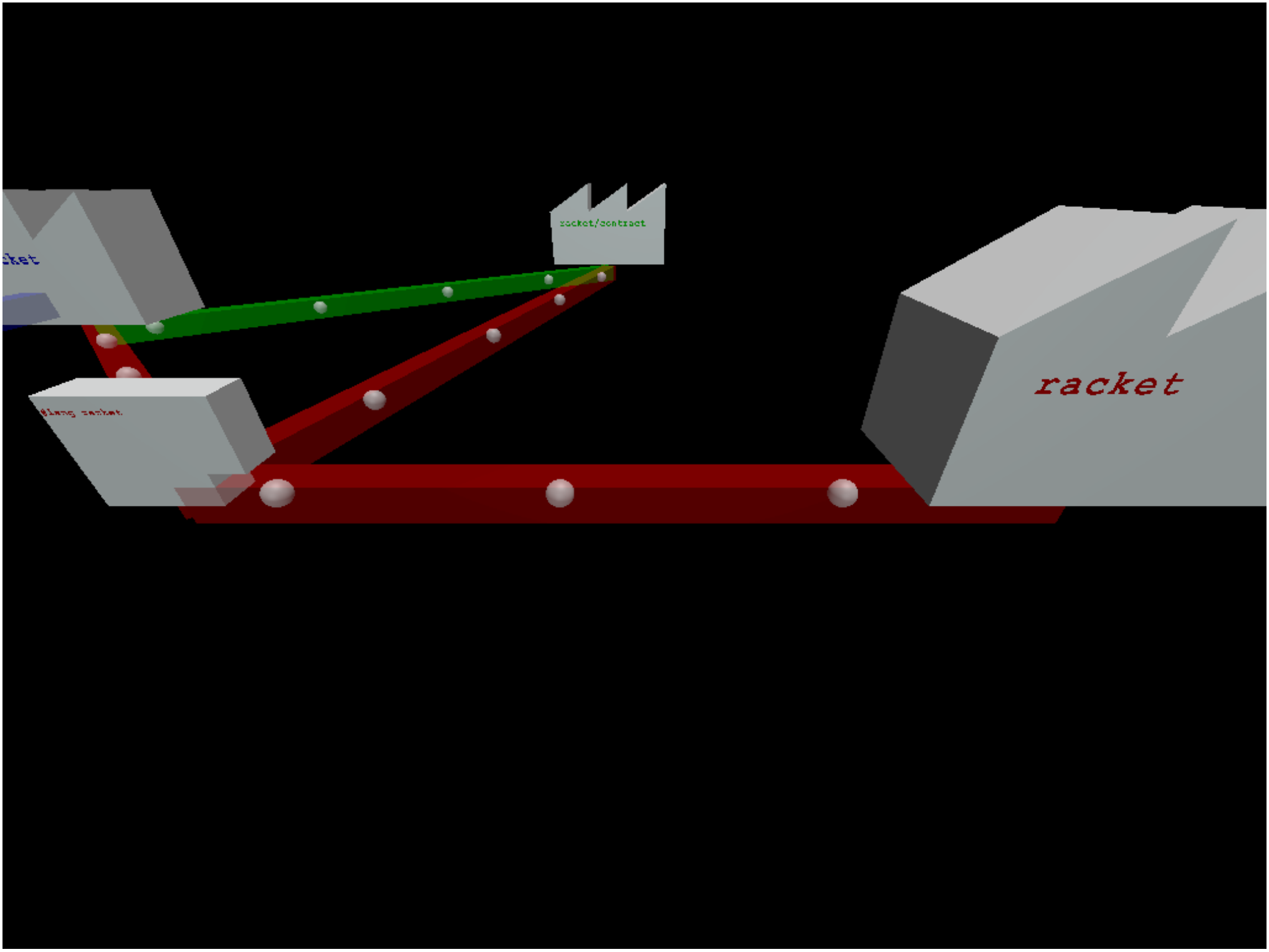

typed/racket

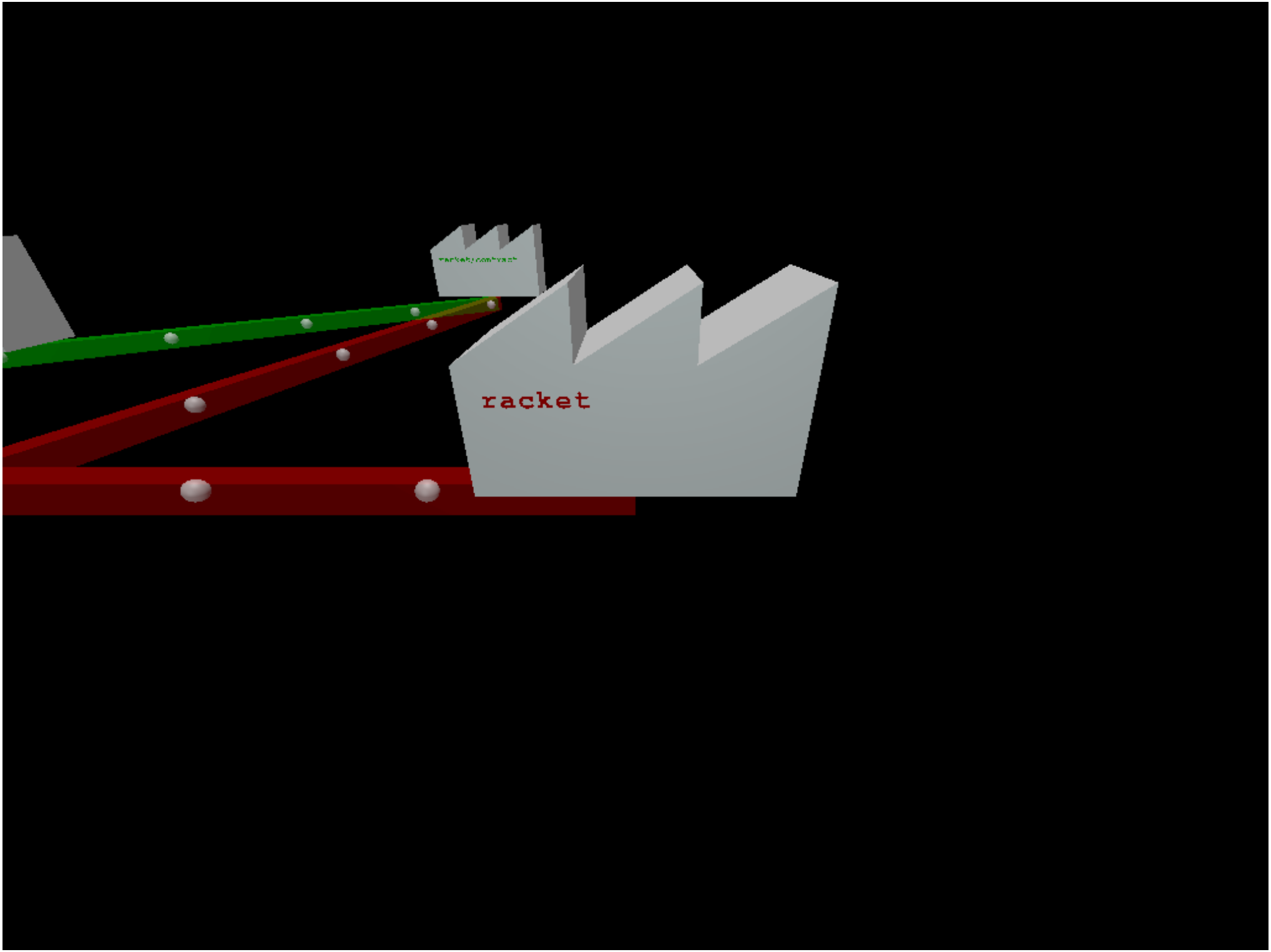

racket/contract

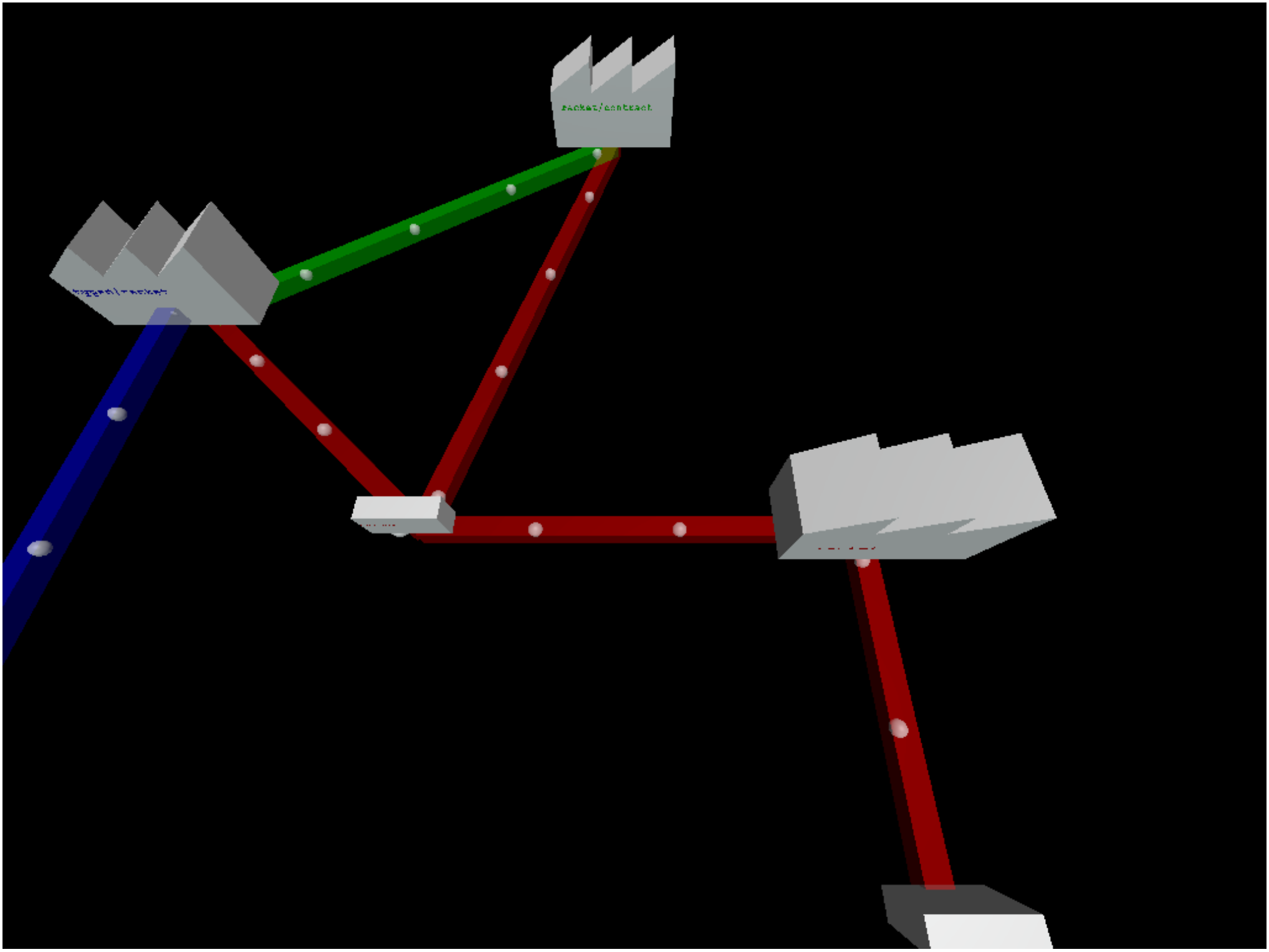

racket/contract

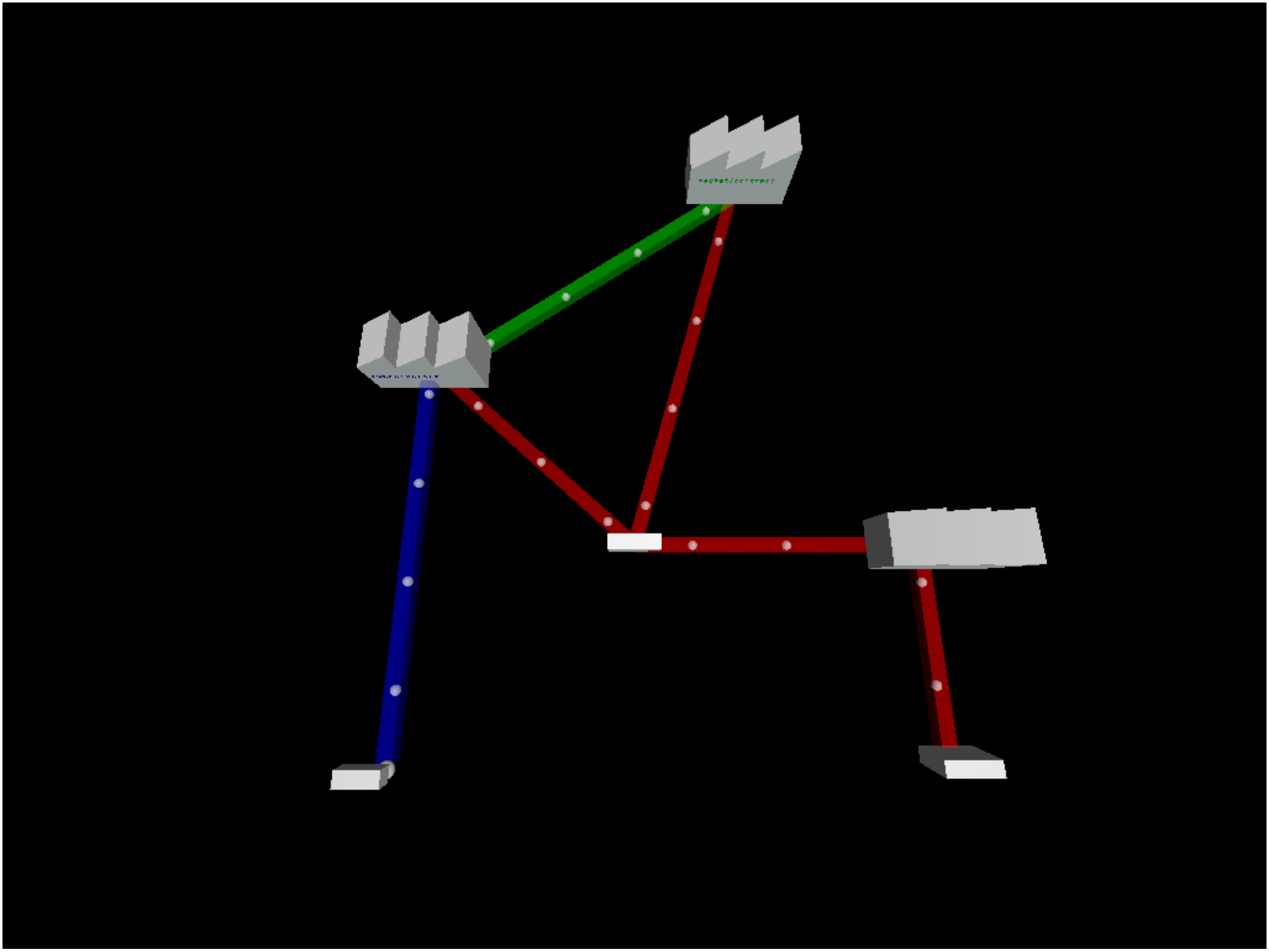

racket/contract

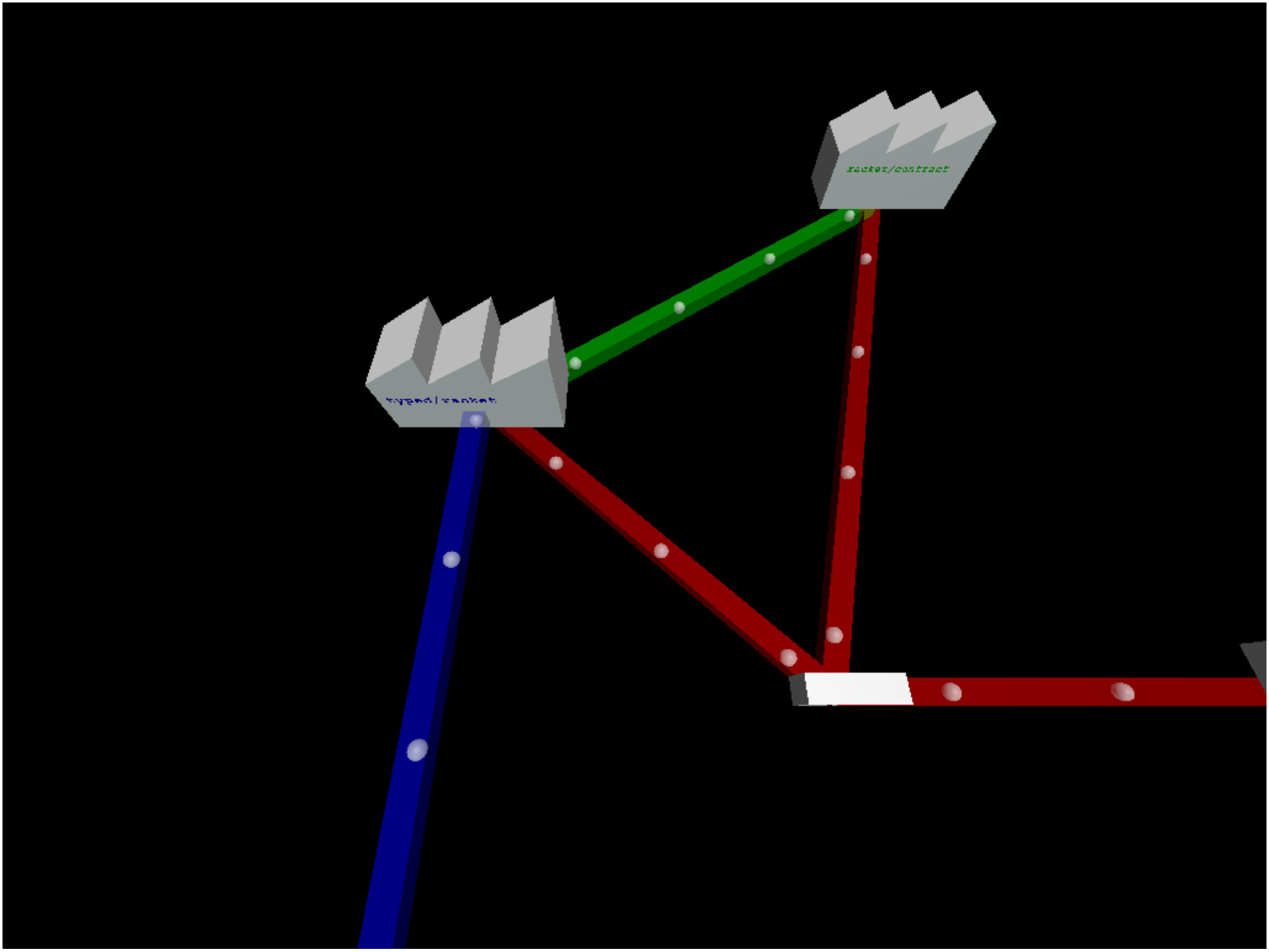

typed/racket
racket/contract
#lang racket

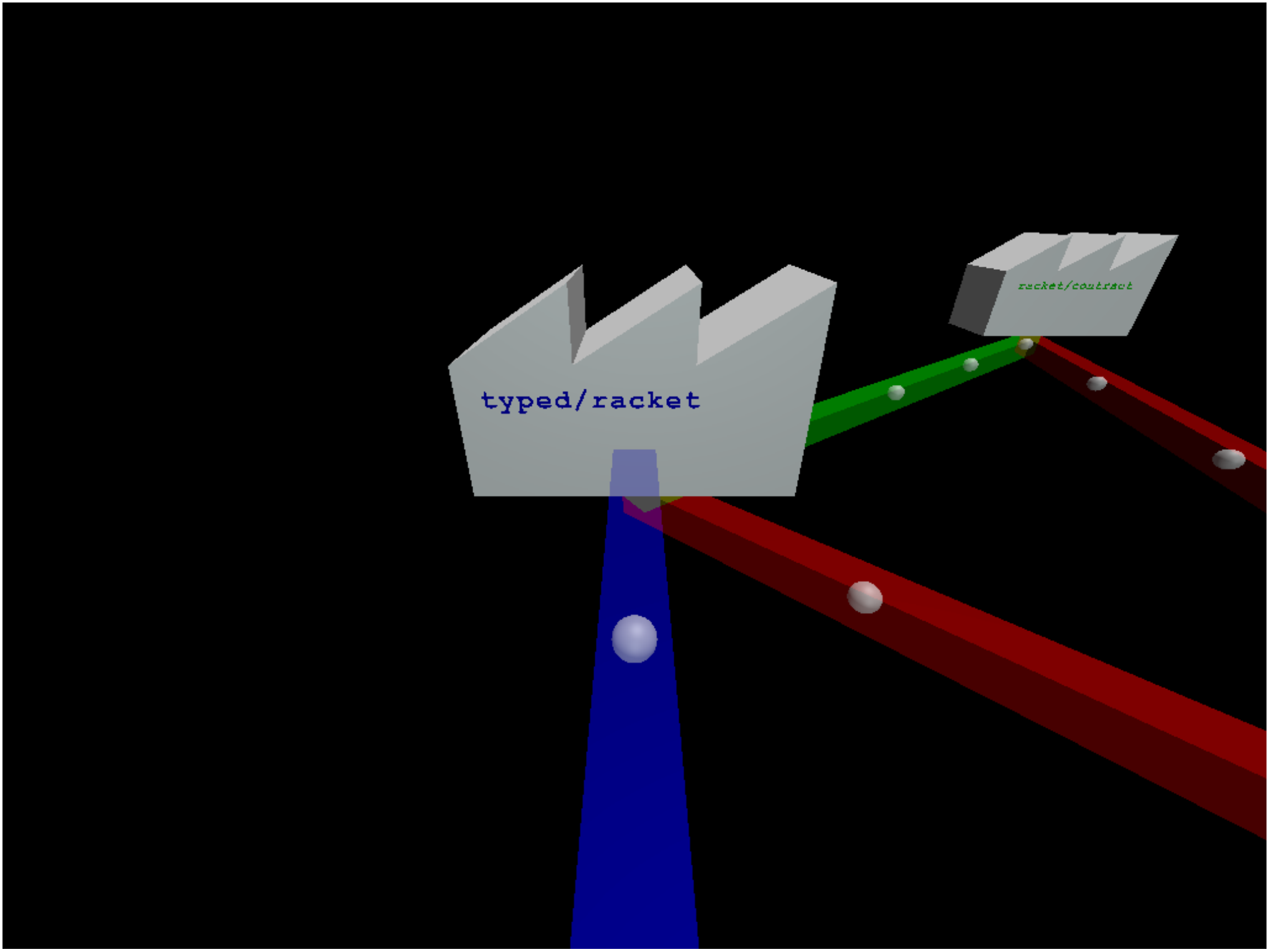

racket/contract
racket

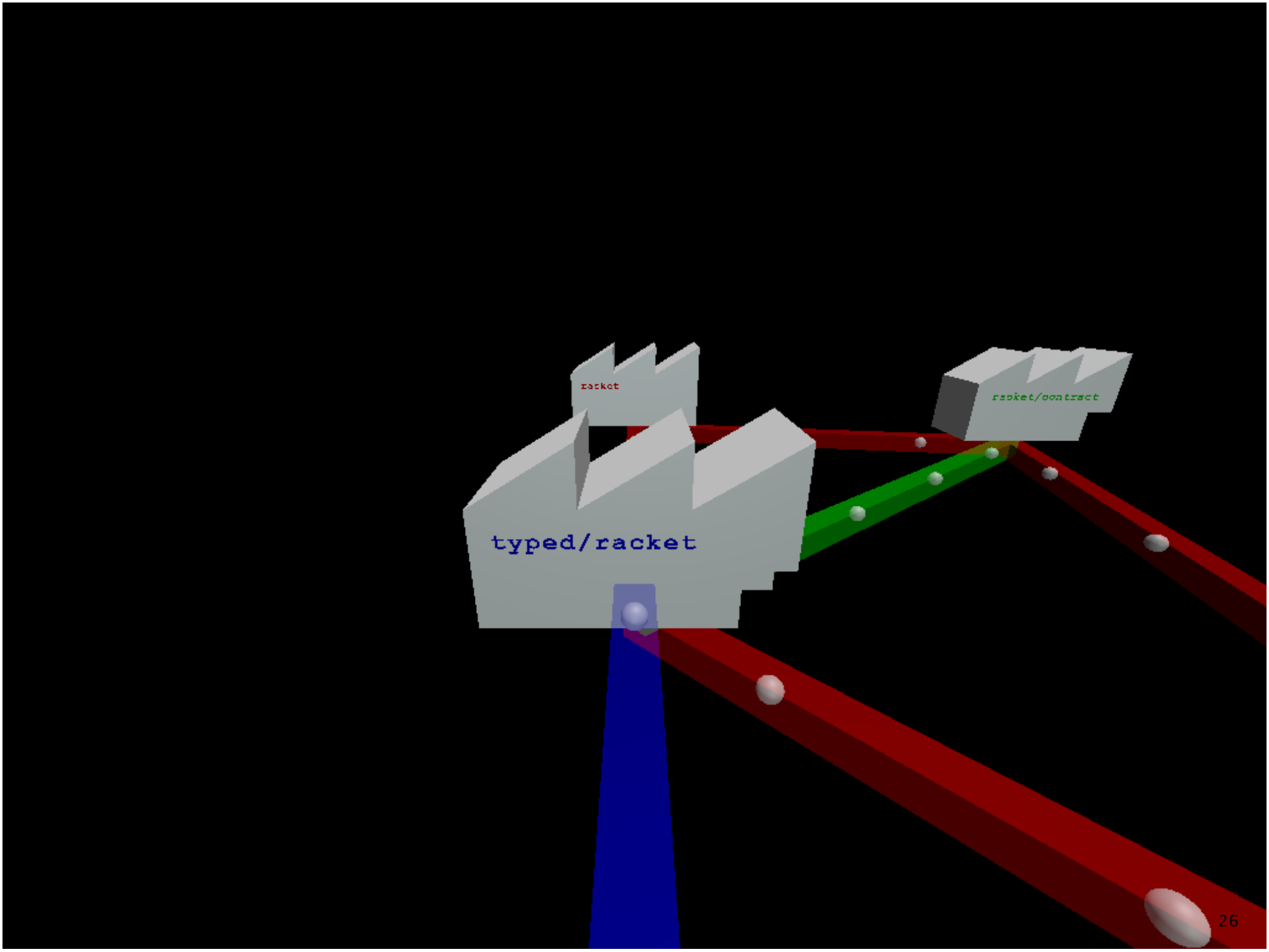

racket

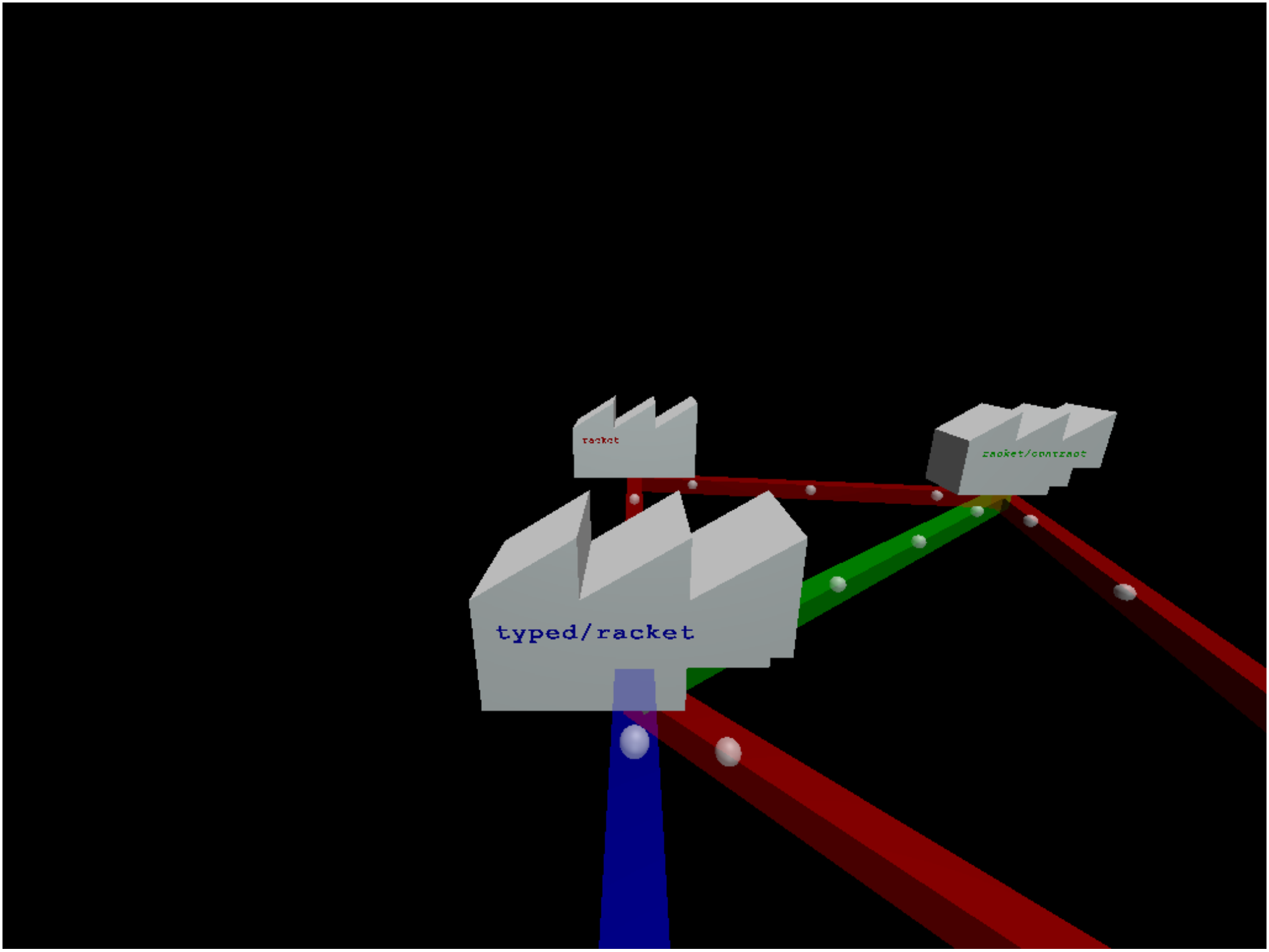

packet/contract

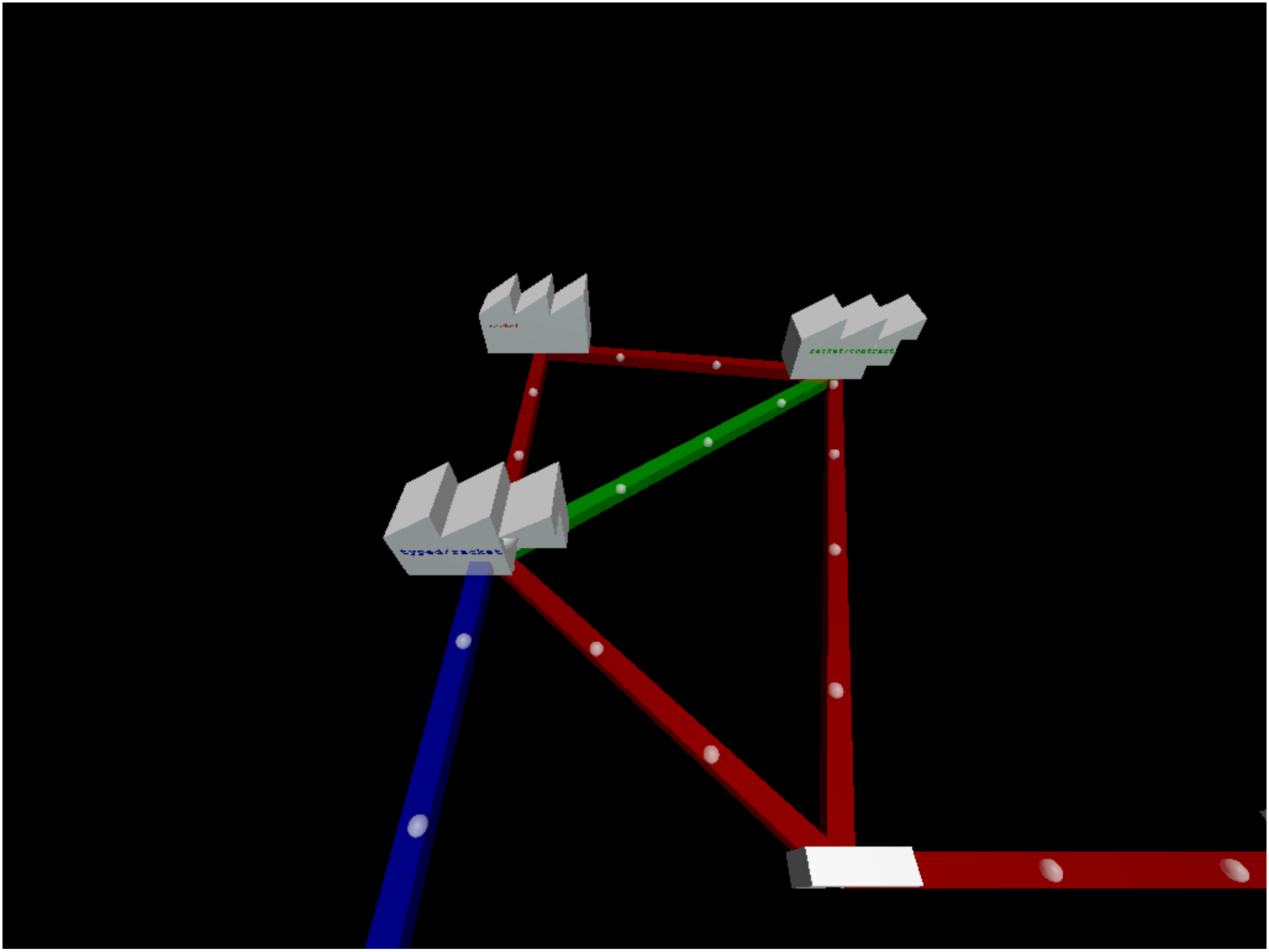


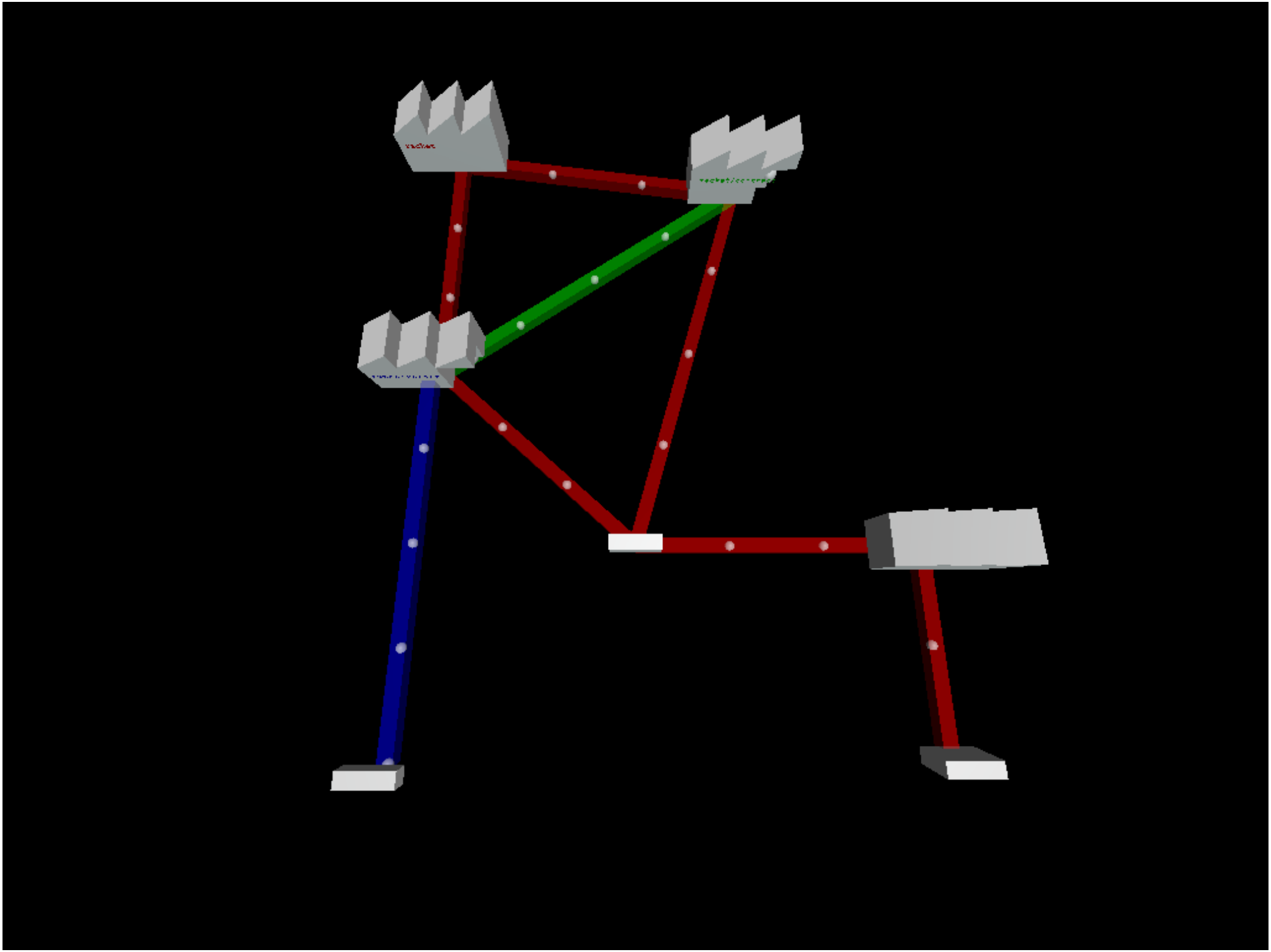

racket/contract
typed/racket

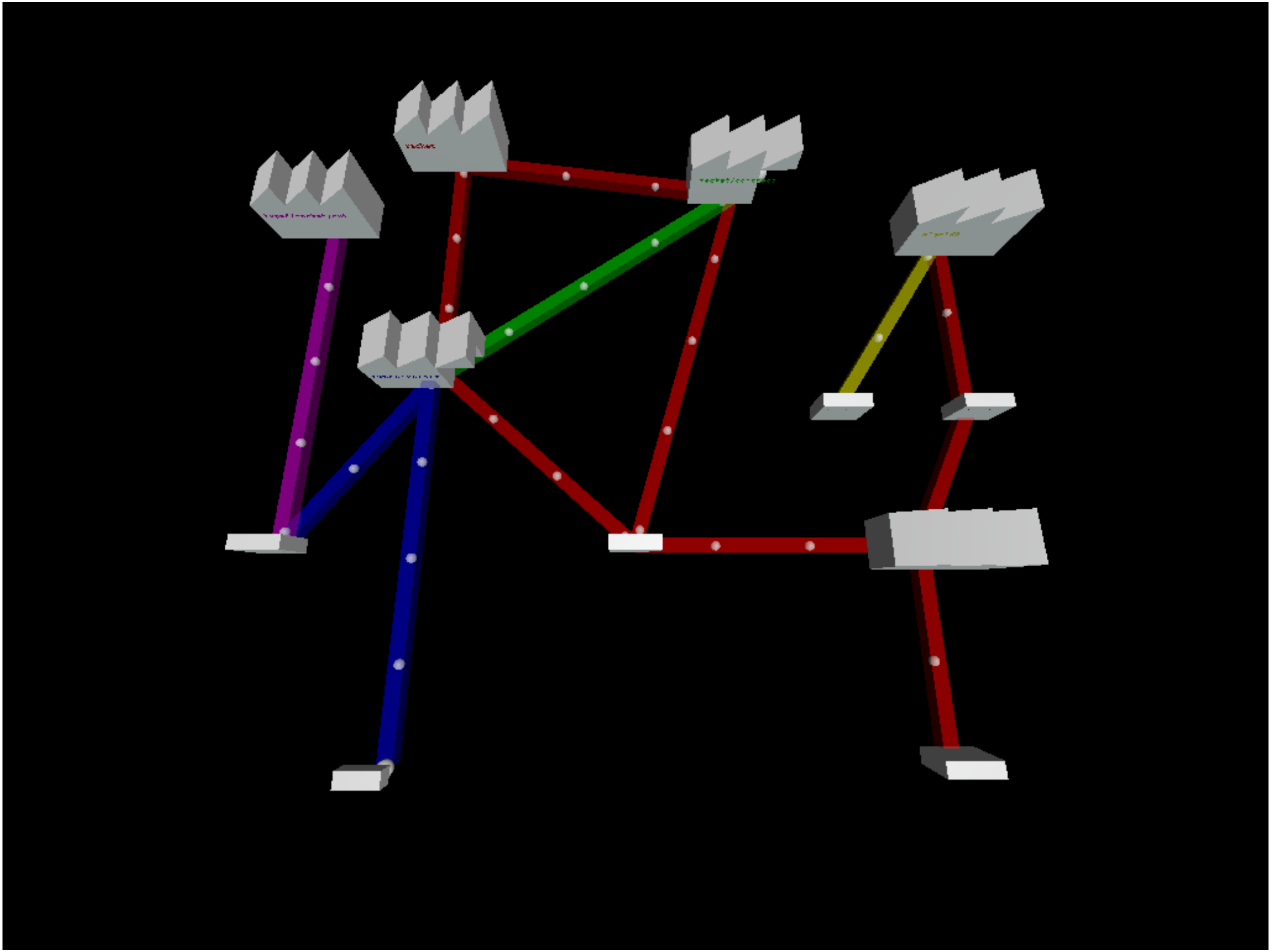

typed/racket
racket/contract

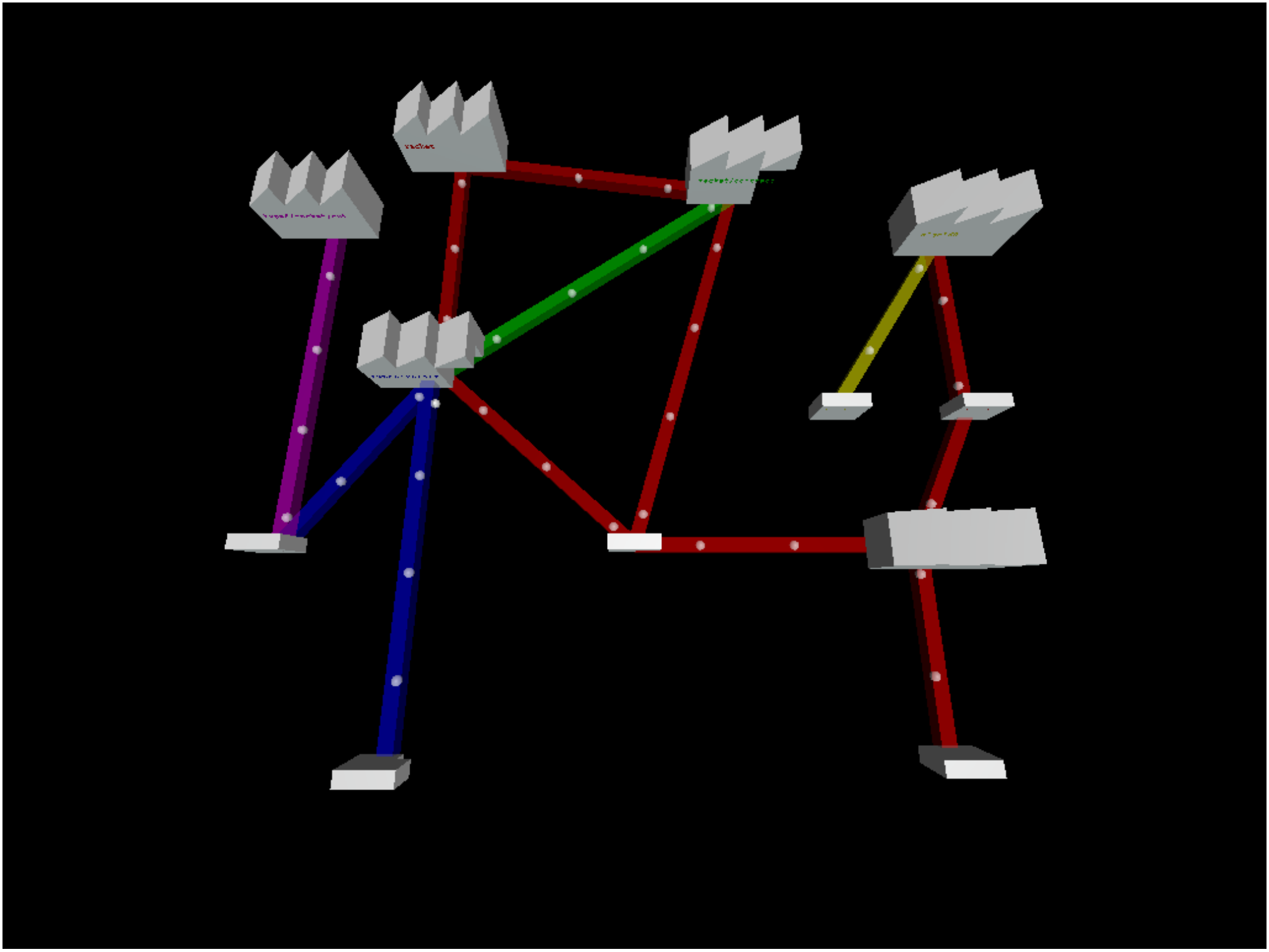

racket
racket/contract
typed/racket

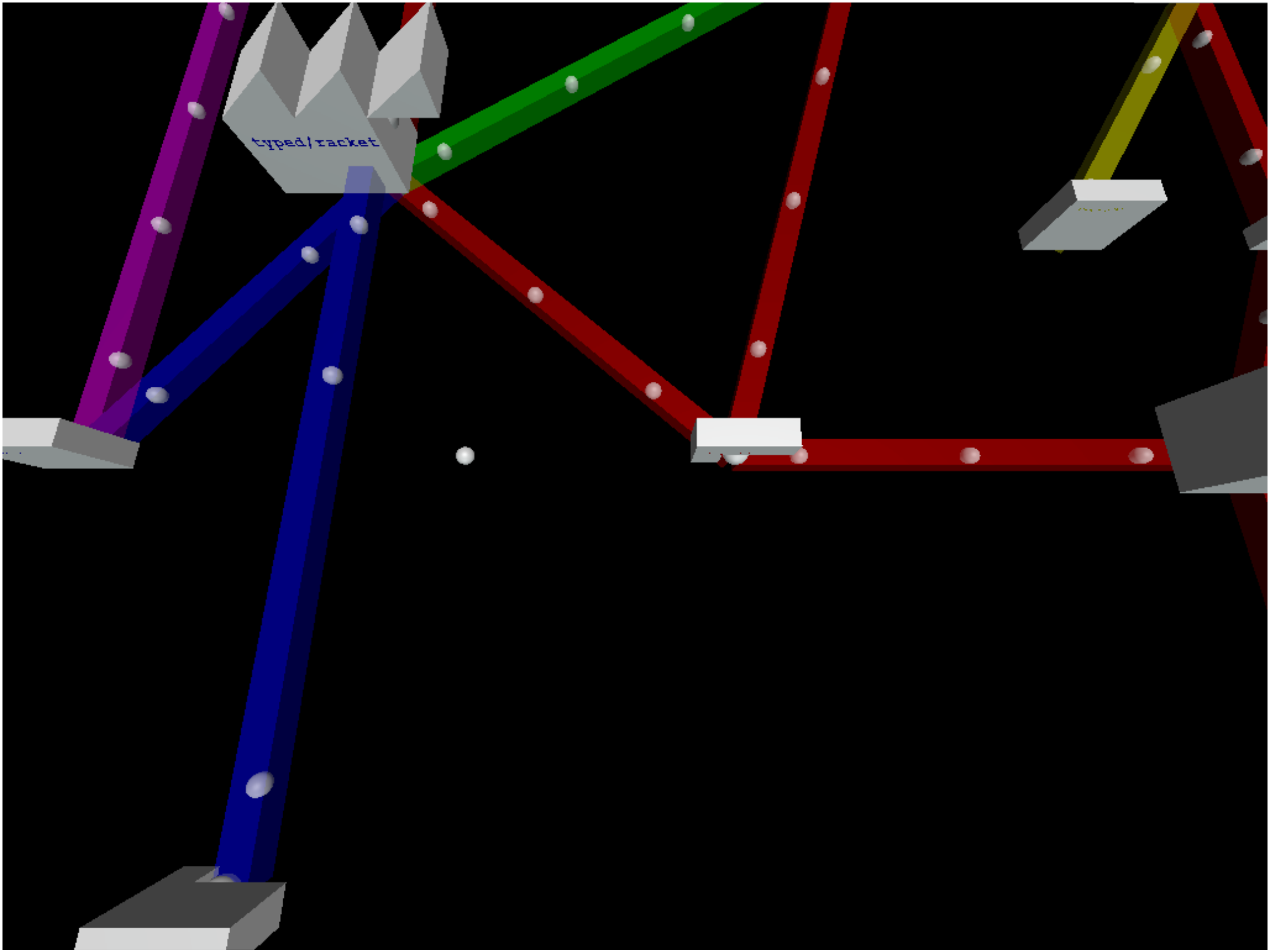

racket
racket/contract
typed/racket

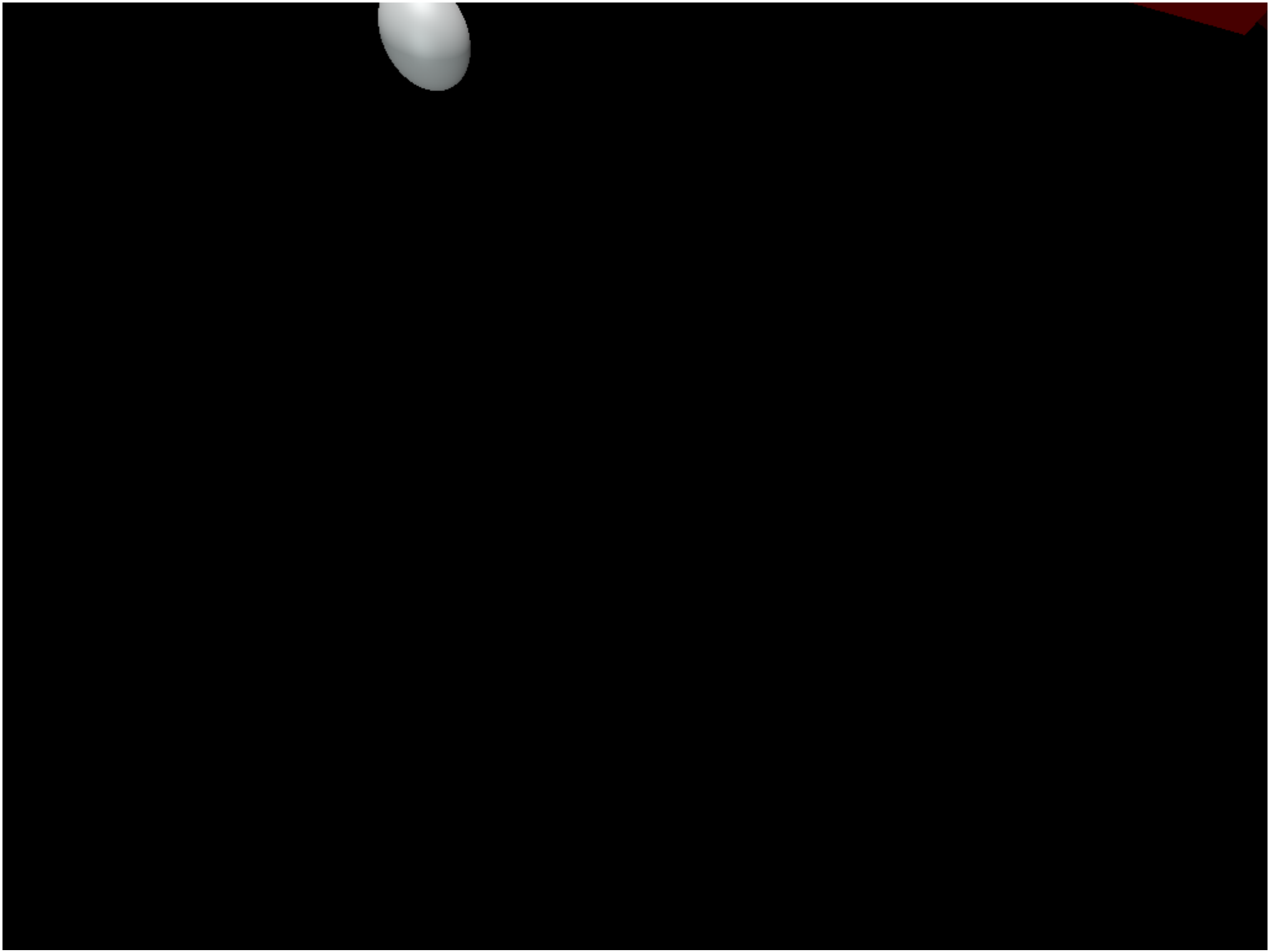

racket/contract
typed/racket

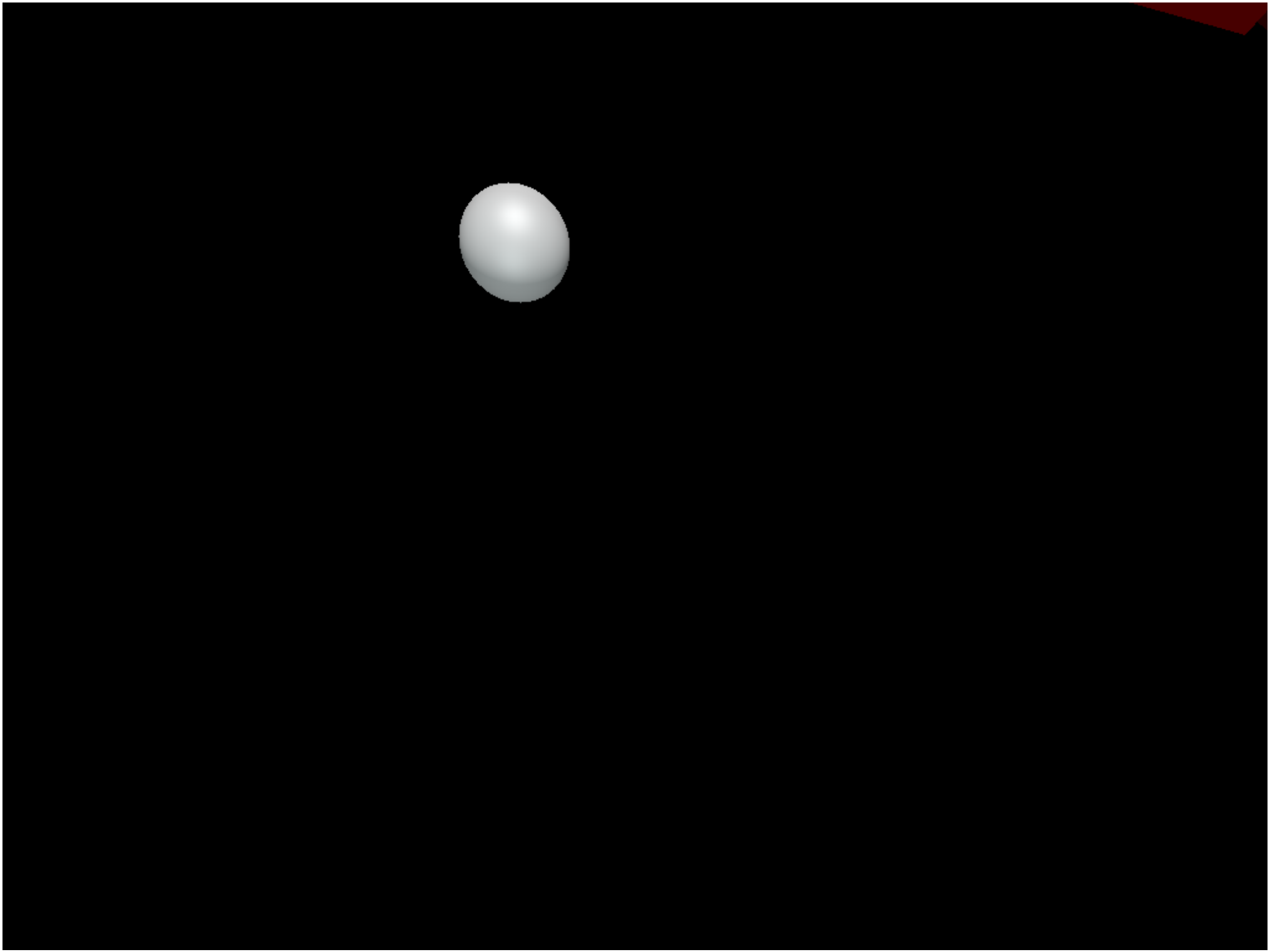


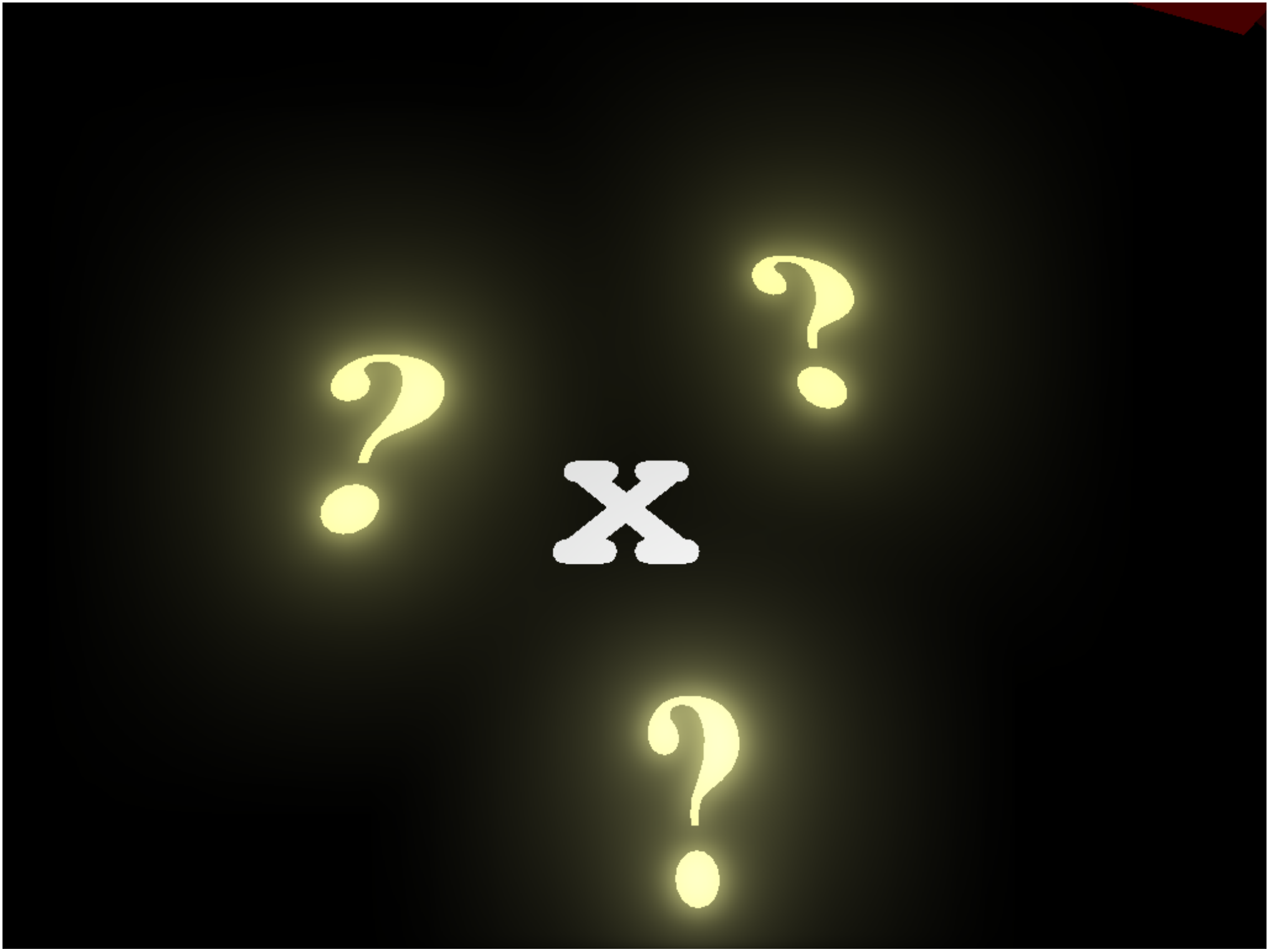





typed/racket



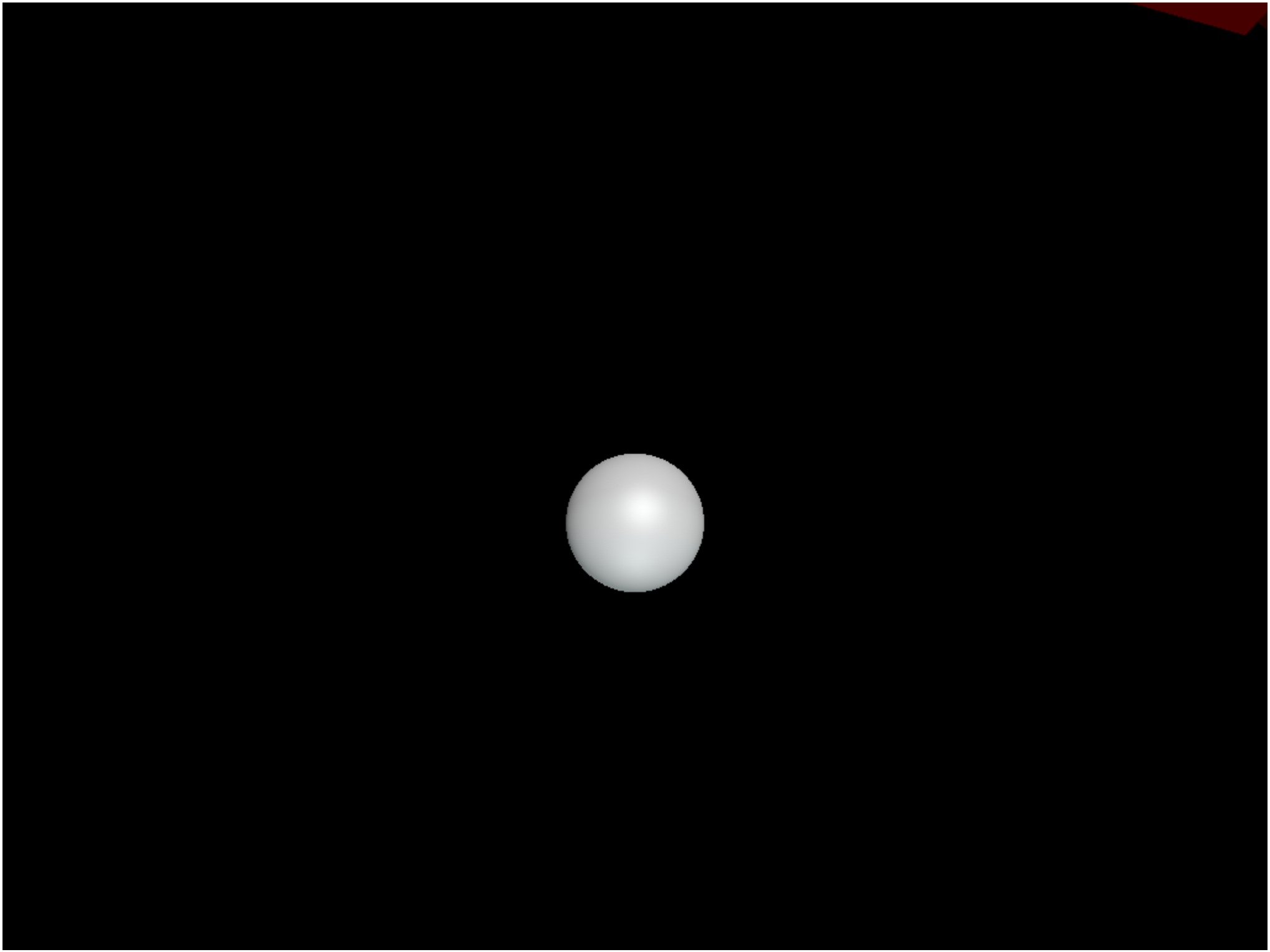


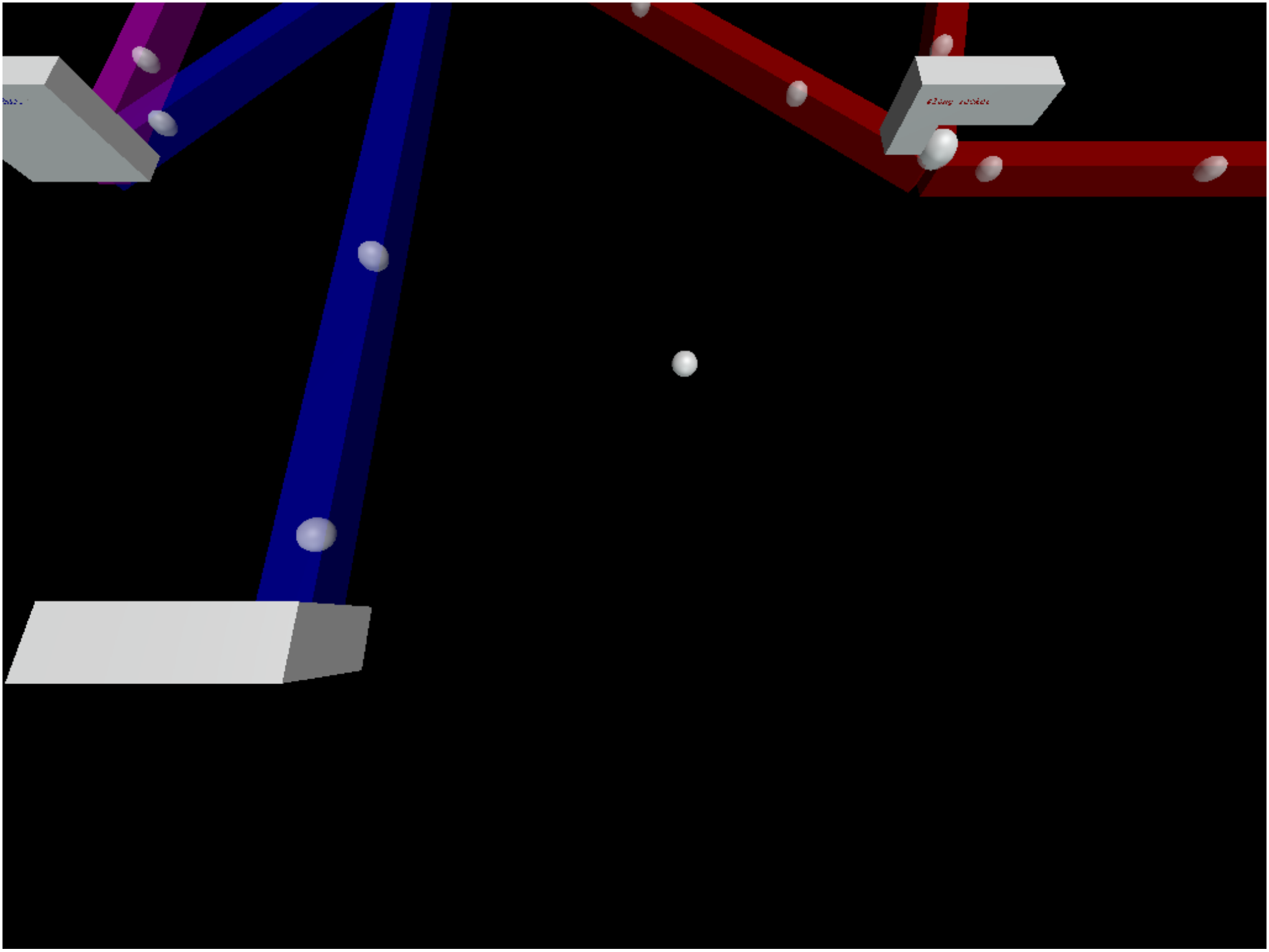

?
?
x
?

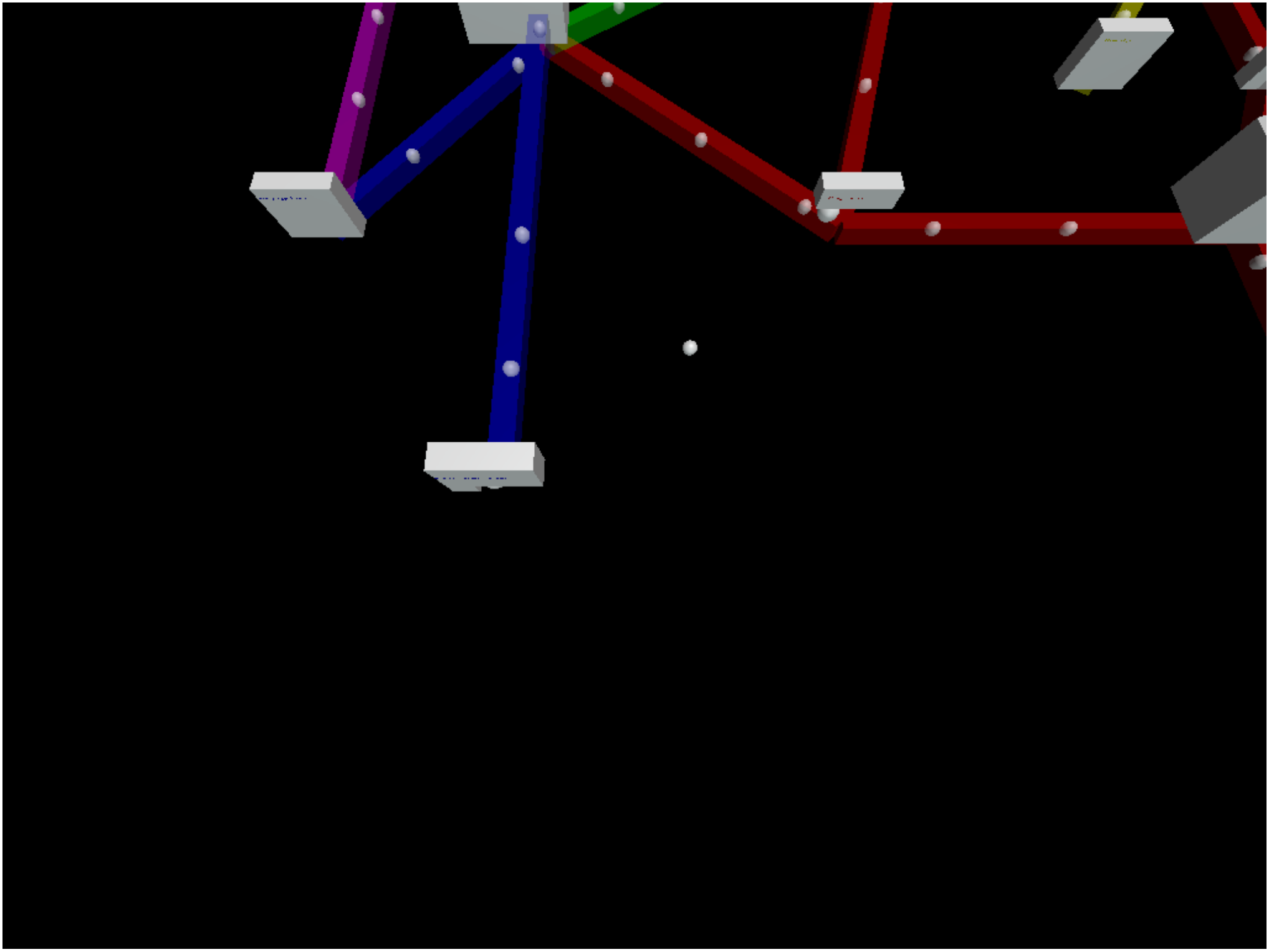
X

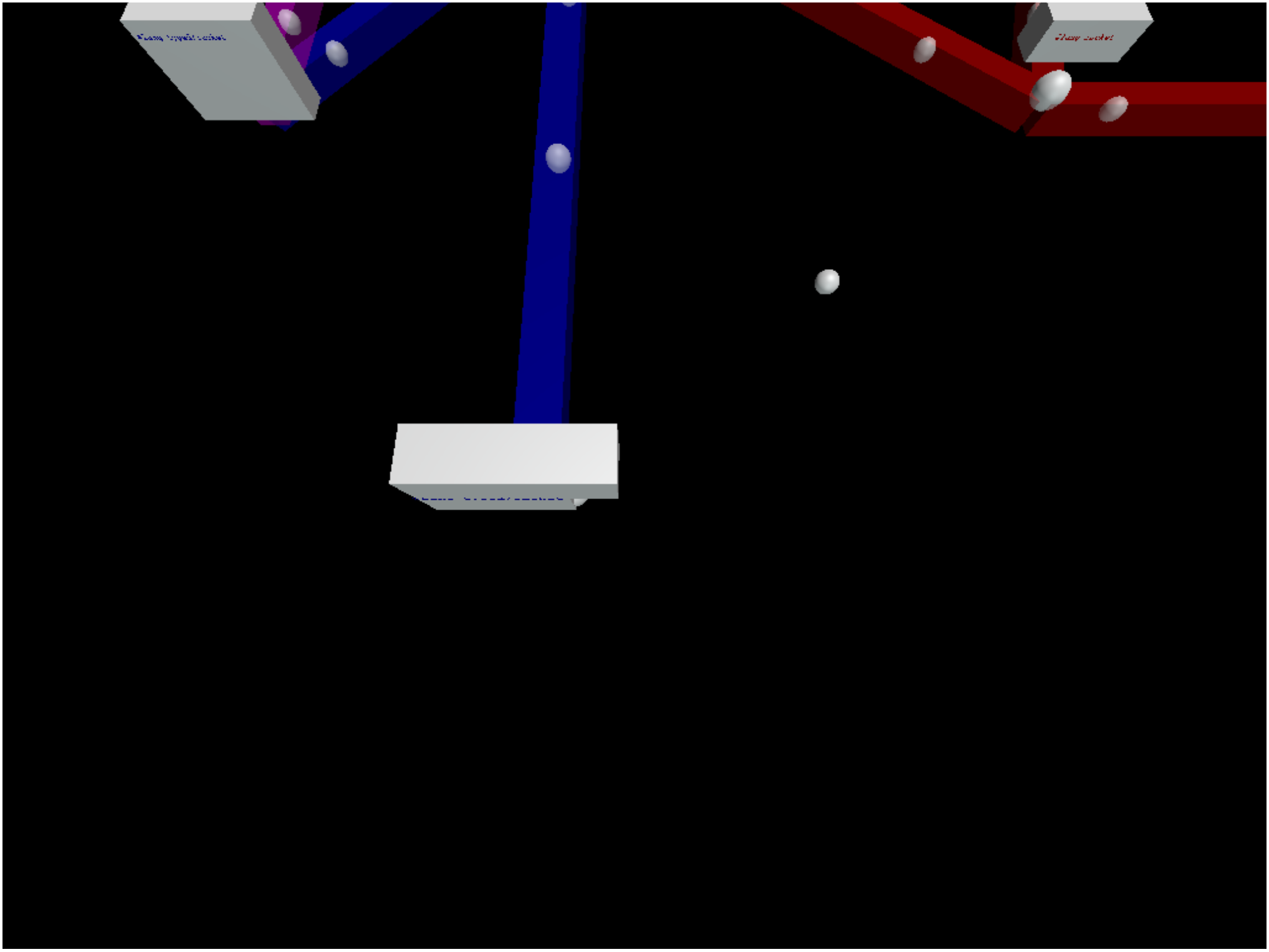
X

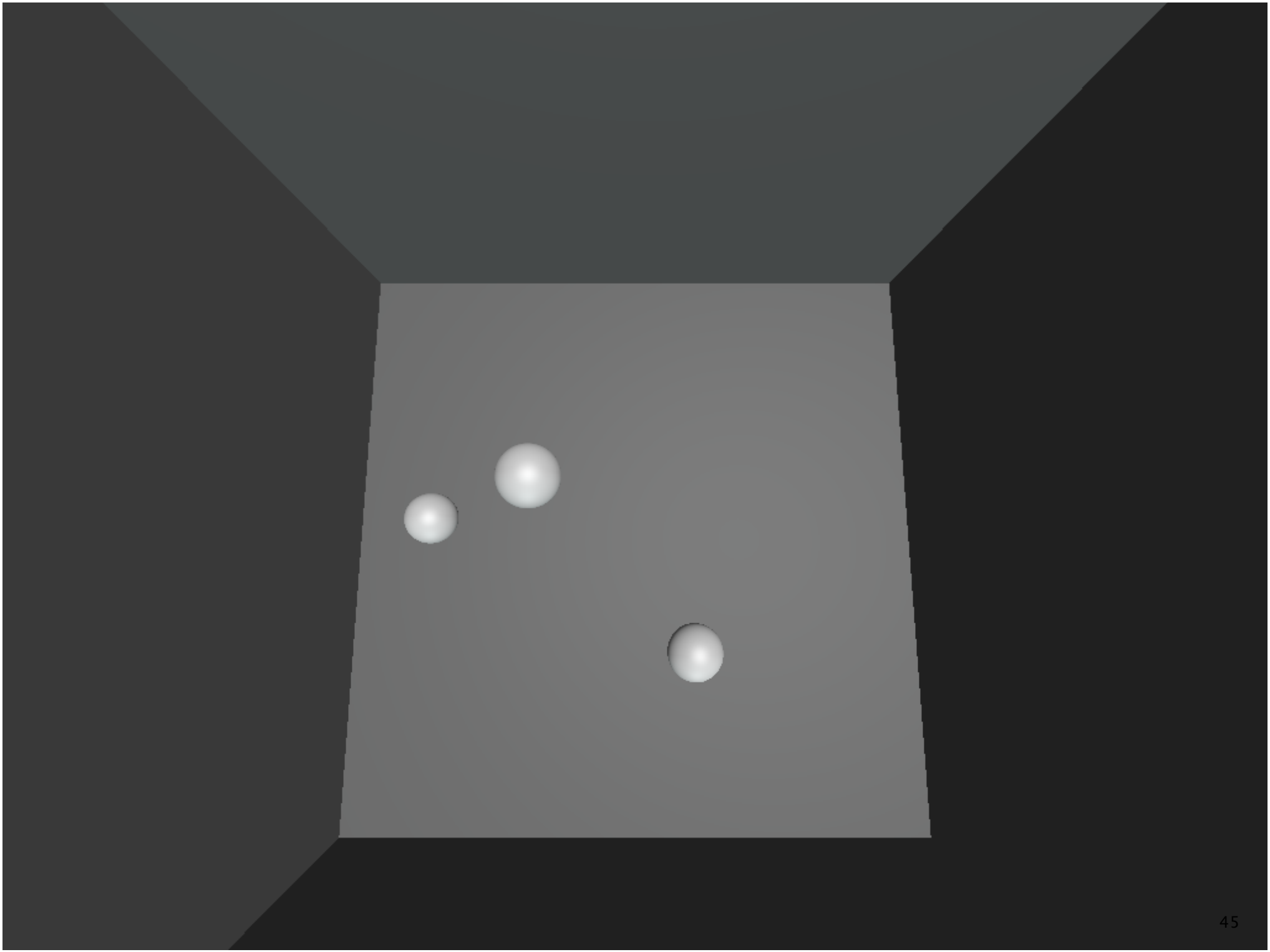
y











```
(define x ....)
```

```
; maybe value for “premade”:
(define x ....)
```

```
; maybe value for “premade”:
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))
```

```
; maybe value for “premade”:
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (premade-or (x)))
```

```
; maybe value for "premade":
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
     ....    ) ; was (premade-or (x))
```

```
; maybe value for "premade":
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
     ....    ) ; was (premade-or (x))
```

```
; maybe value for "premade":
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (      (x))) ; was (premade-or (x))
```

```
; maybe value for "premade":
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (or x (x))) ; was (premade-or (x))
```

```
; maybe value for "premade":
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (or x (x)))
```

```
(define x ....)

(define (f x)
  (define x ....)

  (define-syntax-rule (premade-or e)
    (or x e))

  (let ([x ....])
    (or x (x)))
  ....)

(define y ....)
```

```
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (or x (x)))
```

```
(define-syntax-rule (or a b)
  (let ([x a])
    (if x x b)))

(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (or x (x)))
```

```
(define-syntax-rule (or a b)
  (let ([x a])
    (if x x b)))

(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (let ([x x])
    (if x x (x))))
```

```
(define-syntax-rule (or a b)
  (let ([x a])
    (if x x b)))

(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (let ([x x])
    (if x x (x))))
```

```
(define-syntax-rule (or a b)
  (let ([x a])
    (if x x b)))
(define x ....)
(define-syntax-rule (premade-or e)
  (or x e))
(let ([x ....])
  x
    x   )
(let ([x   ])
  (if x x ( )))
```

```
(define-syntax-rule (or a b)
  (let ([x a])
    (if x x b)))

(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (let ([x x])
    (if x x (x))))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (or y x)))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (or y x)))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (or y x)))
```

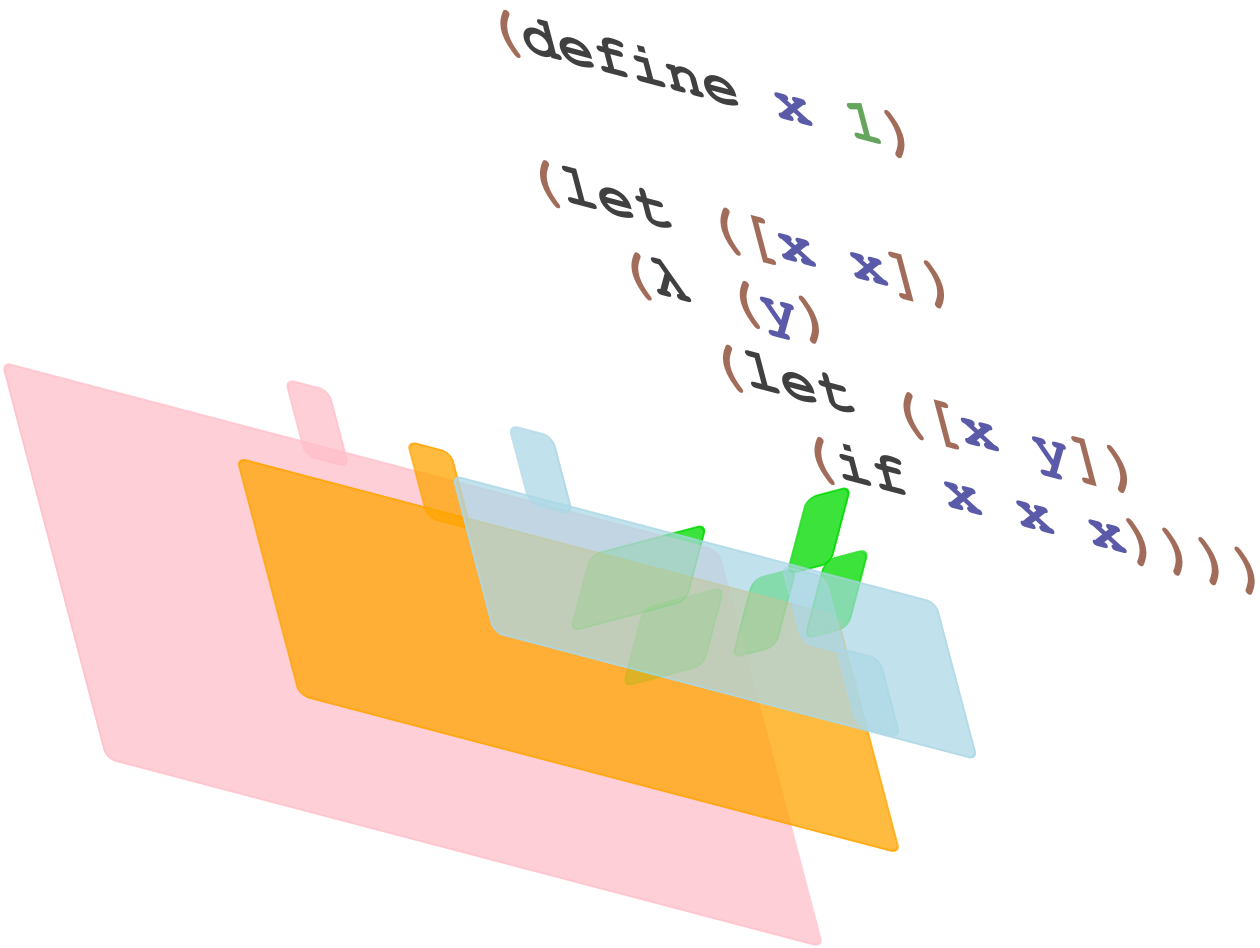
```
(define x 1)

(let ([x x])
  (λ (y)
    (or y x)))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (or y x)))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (or y x)))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (let ([x y])
      (if x x x))))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (let ([x y])
      (if x x x))))
```


(define x 1)
(let ([x x])
(λ (y)
(let ([x y])
(if x x x))))

```
(define x 1)

(let ([x x])
  (λ (y)
    (let ([x y])
      (if x x x))))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (let ([x y])
      (if x x x))))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (let ([x y])
      (if x x x))))
```

```
(define x 1)

(let ([x x])
  (λ (y)
    (let ([x y])
      (if x x x))))
```

```
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (premade-or (x)))
```

```
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (premade-or (x)))
```

```
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
     ....    )  ; was (premade-or (x))
```

```
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
     ....    ) ; was (premade-or (x))
```

```
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (      (x))) ; was (premade-or (x))
```

```
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (or x (x))) ; was (premade-or (x))
```

```
(define x ....)

(define-syntax-rule (premade-or e)
  (or x e))

(let ([x ....])
  (or x (x)))
```

= scope

= scope

= scope set

= scope

= scope set

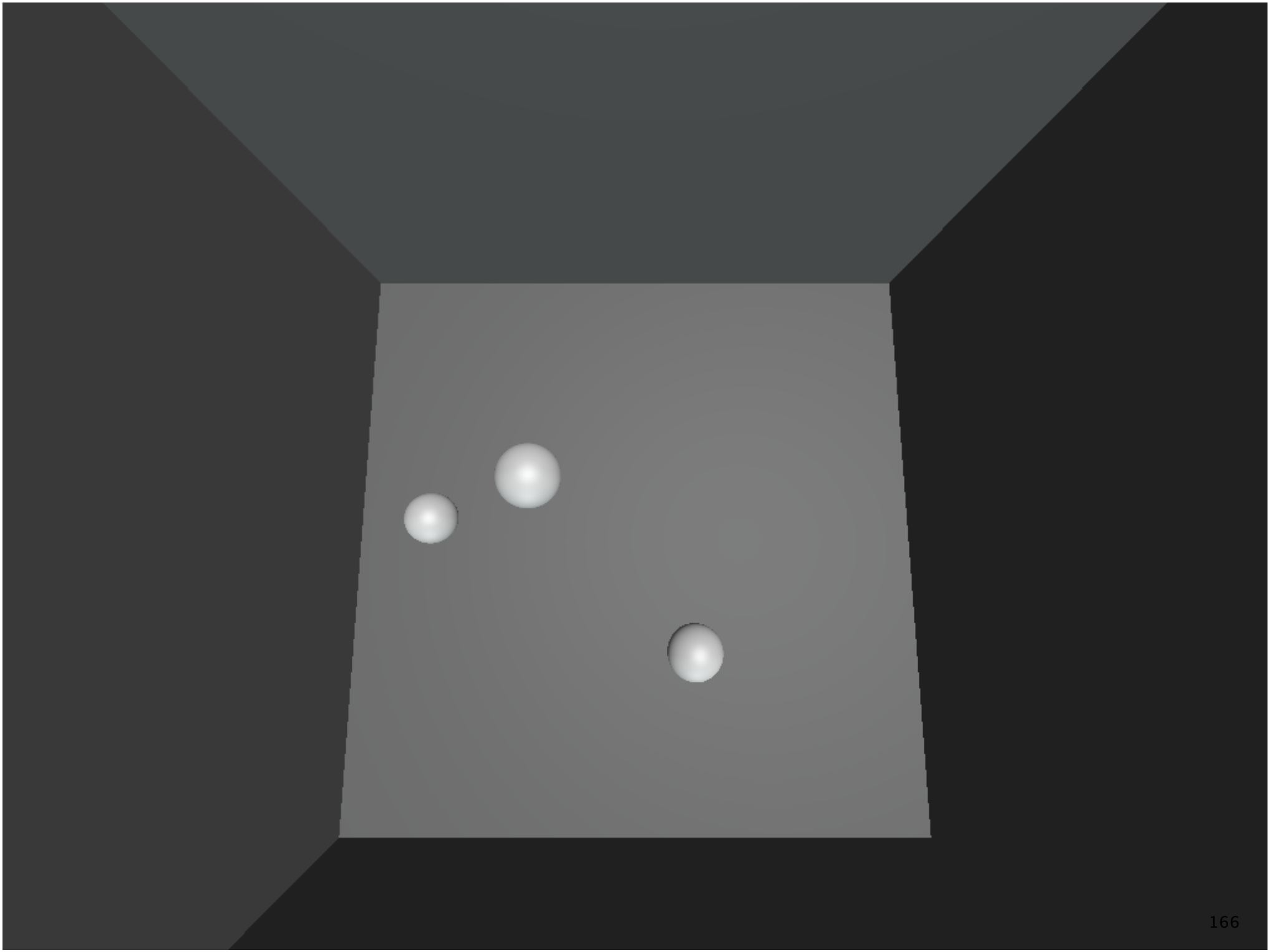
= syntax object

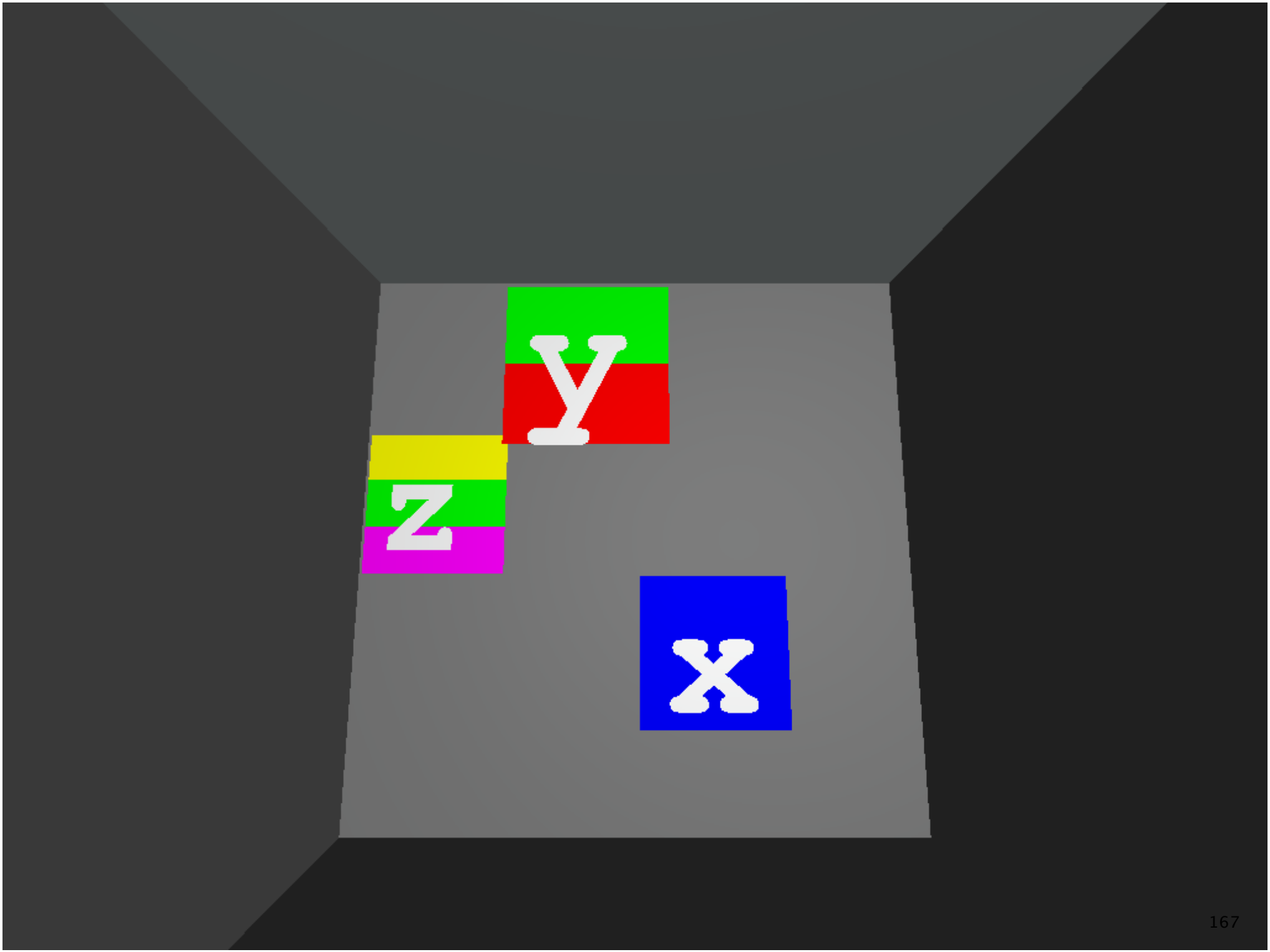
= scope

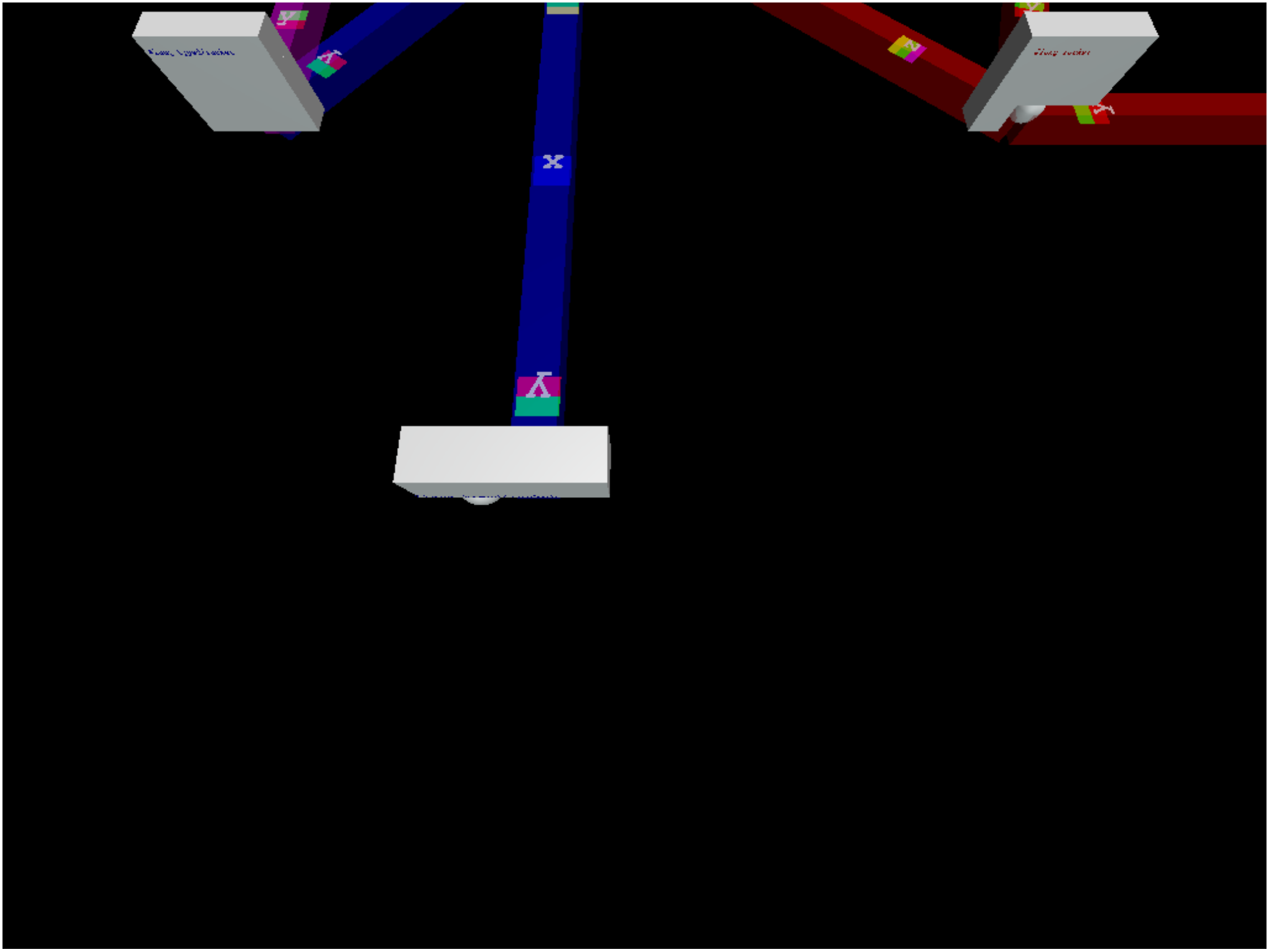
= scope set

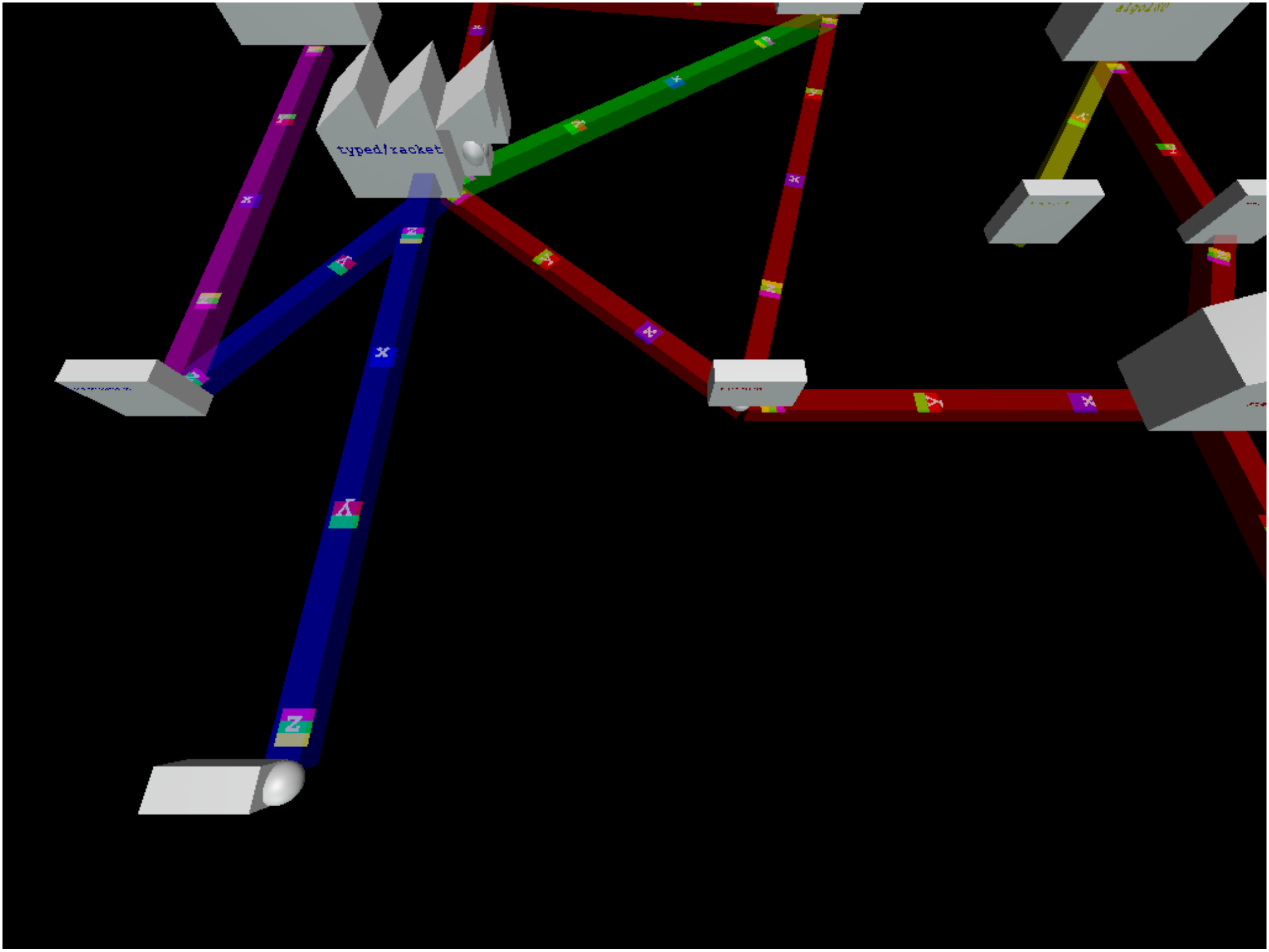
= syntax object

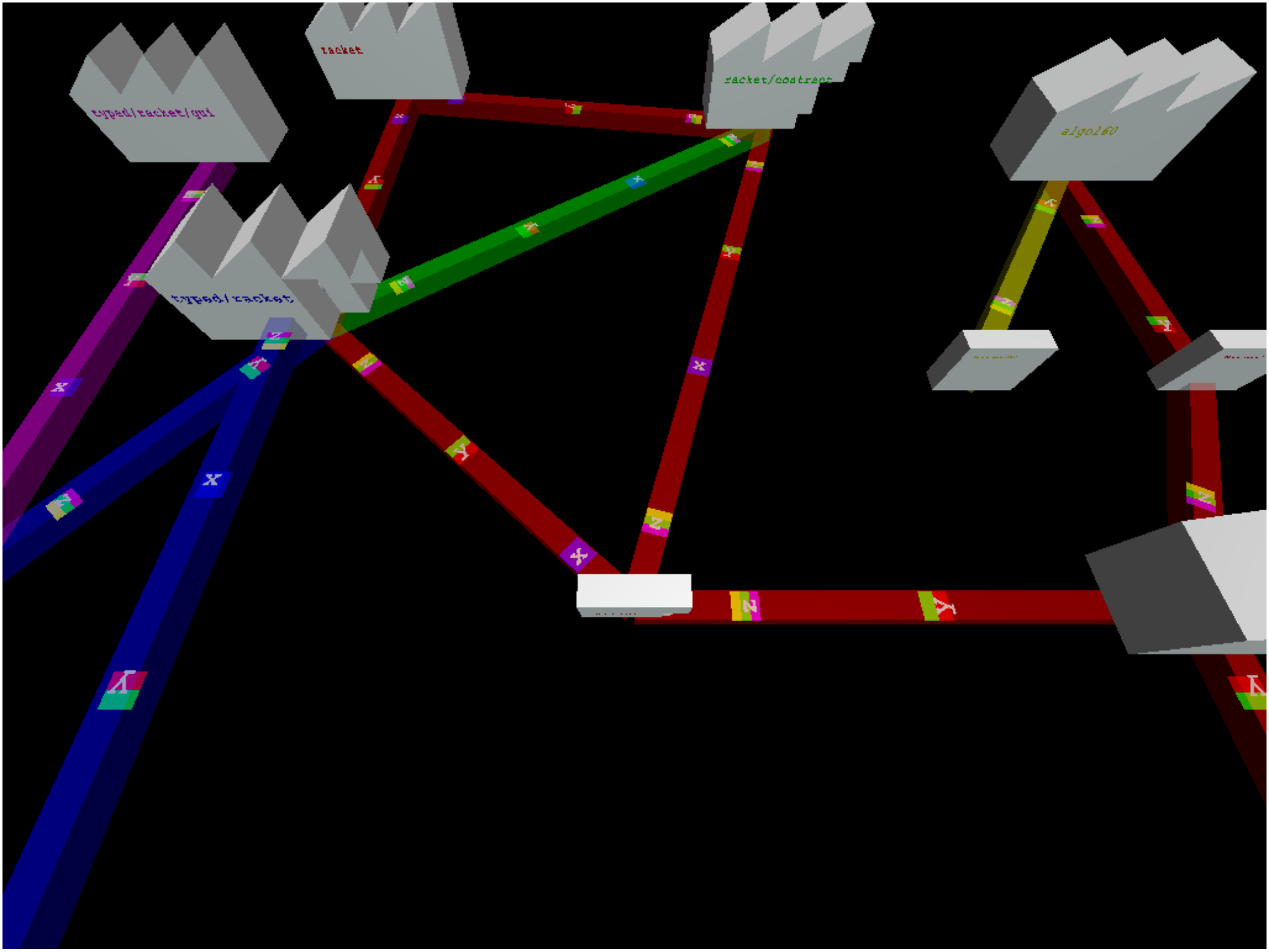
**http://www.cs.utah.edu/~mflatt/scope-sets-5/**

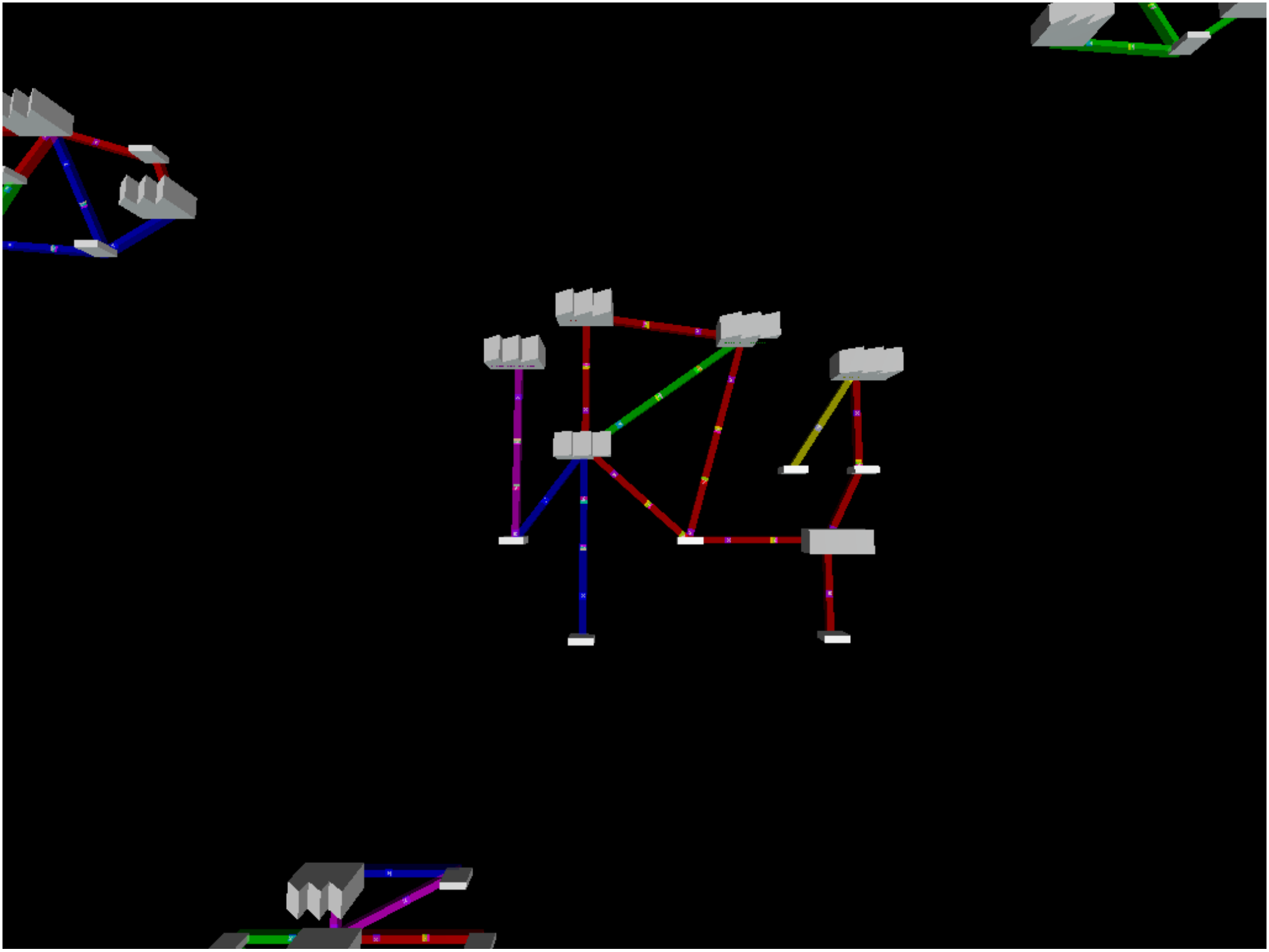


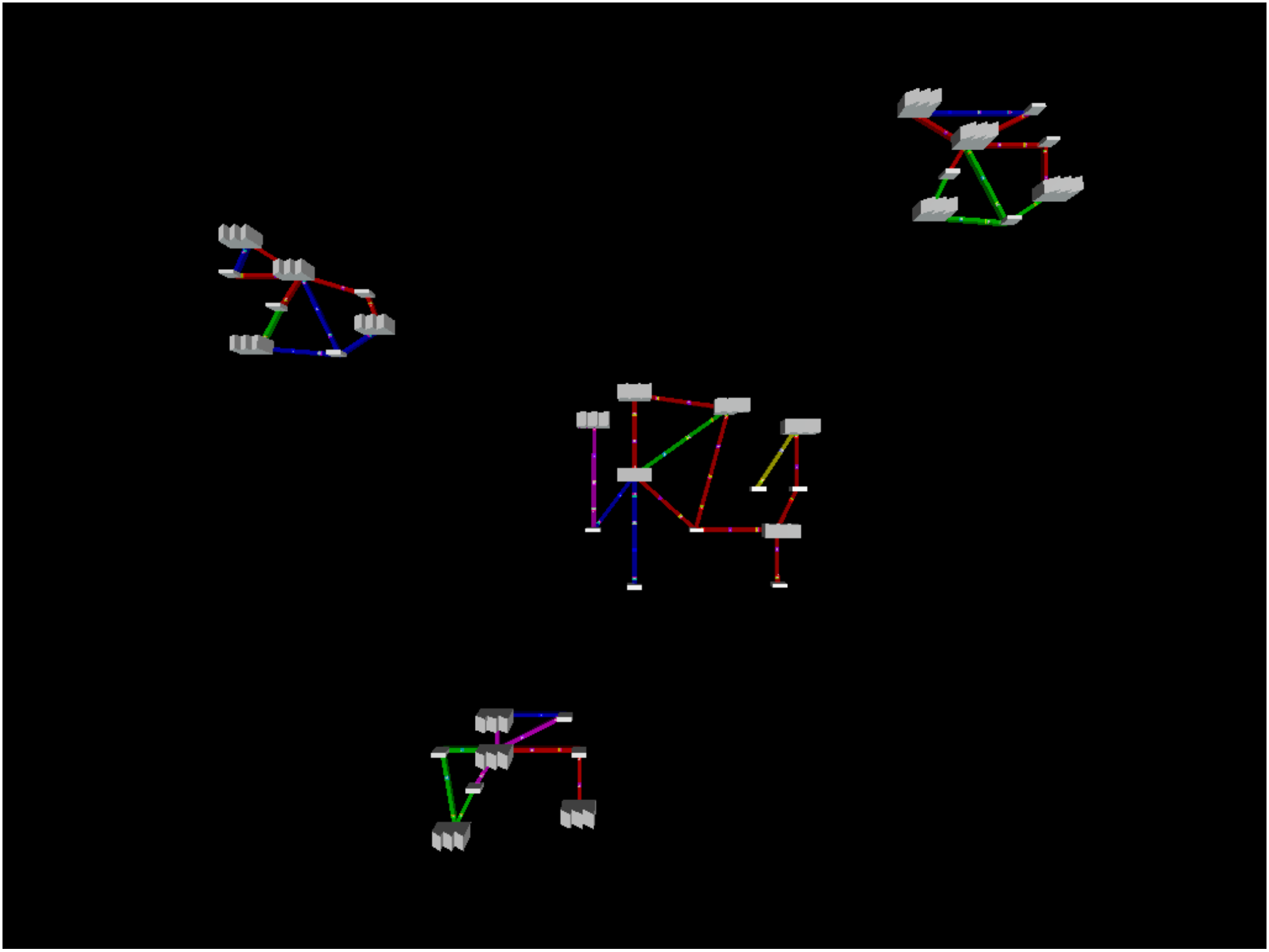

y
z
x

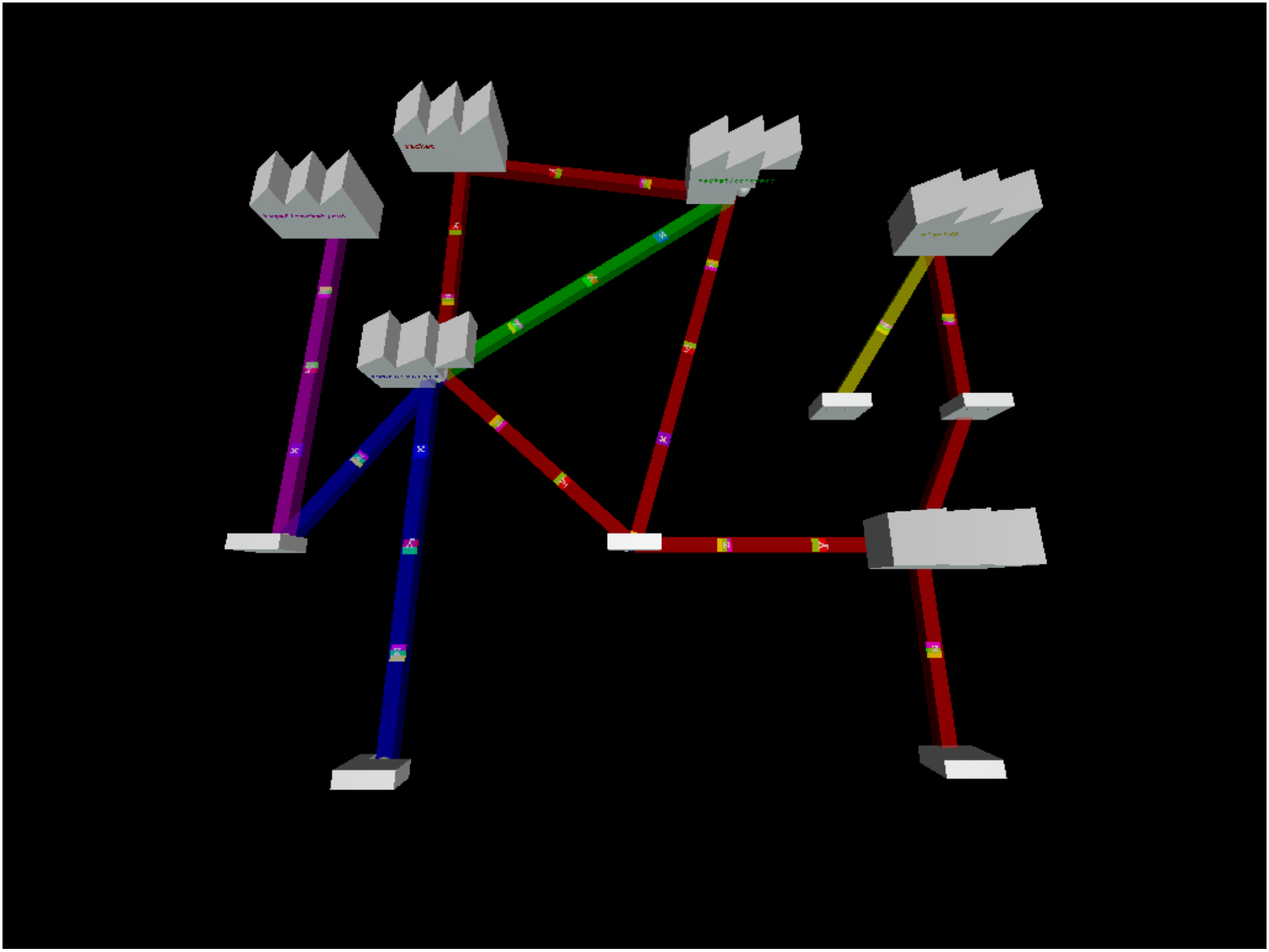

x
y

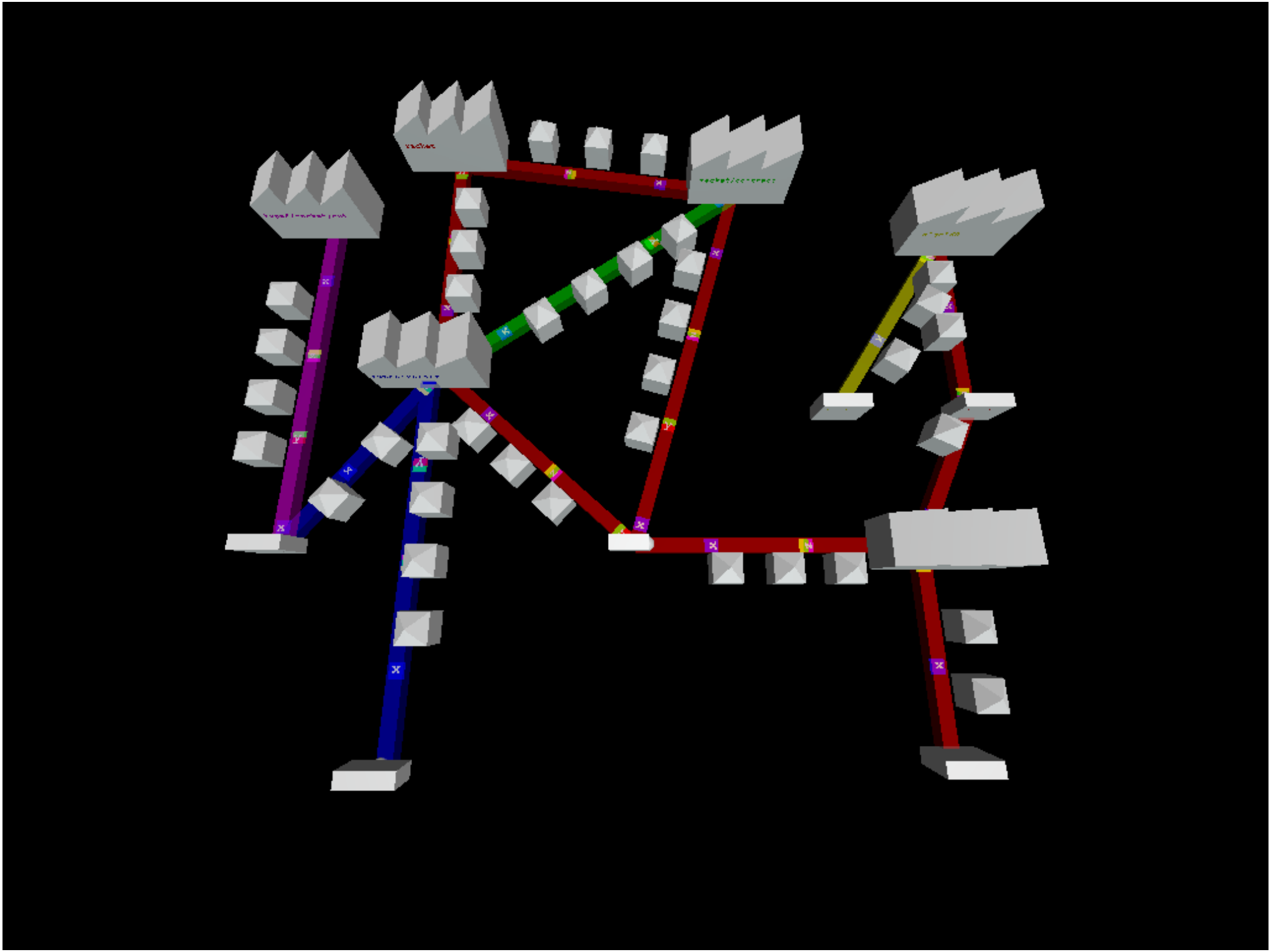

typed/racket

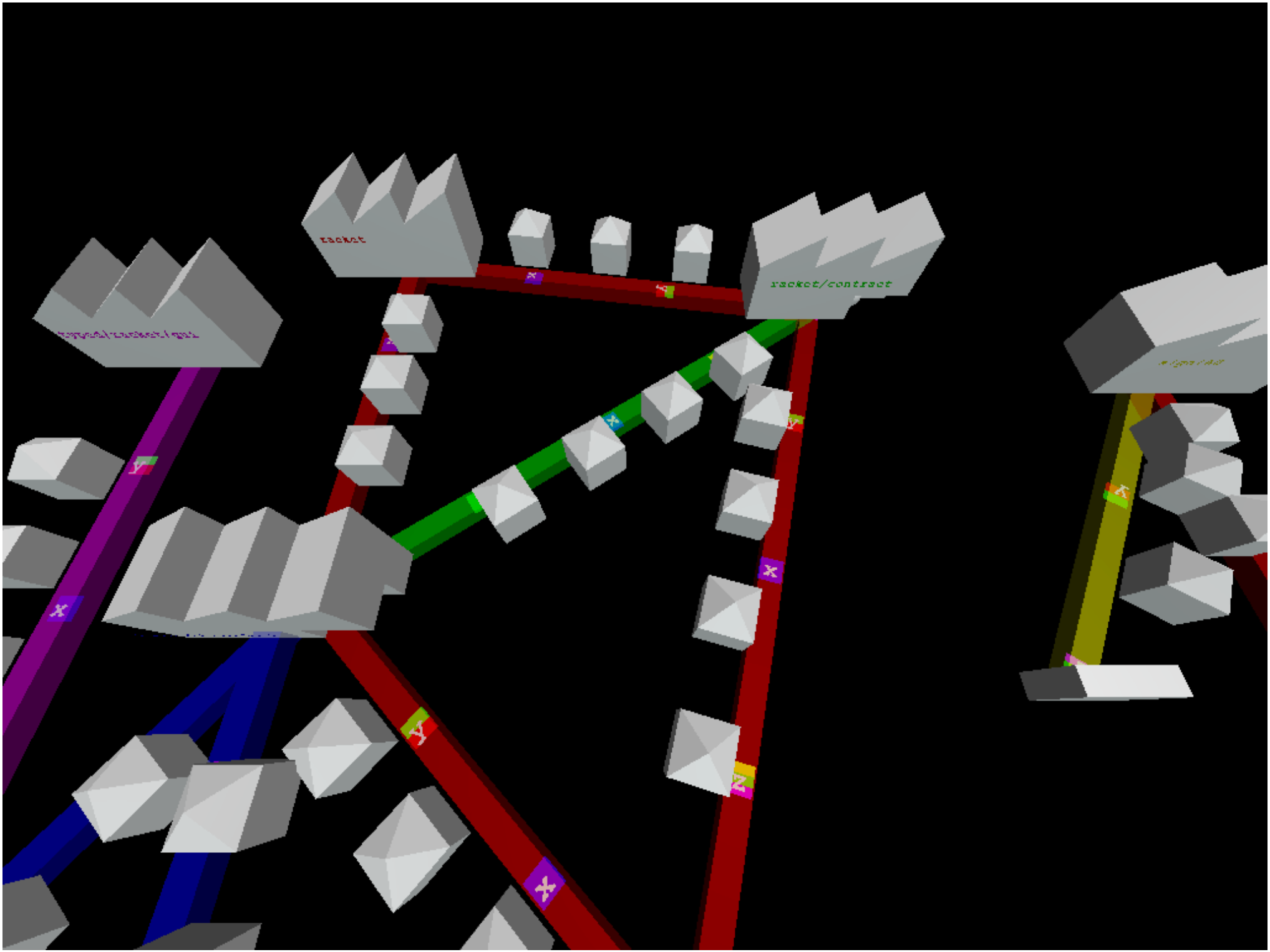

typed/racket/gui
racket
racket/contract
algol60
typed/racket

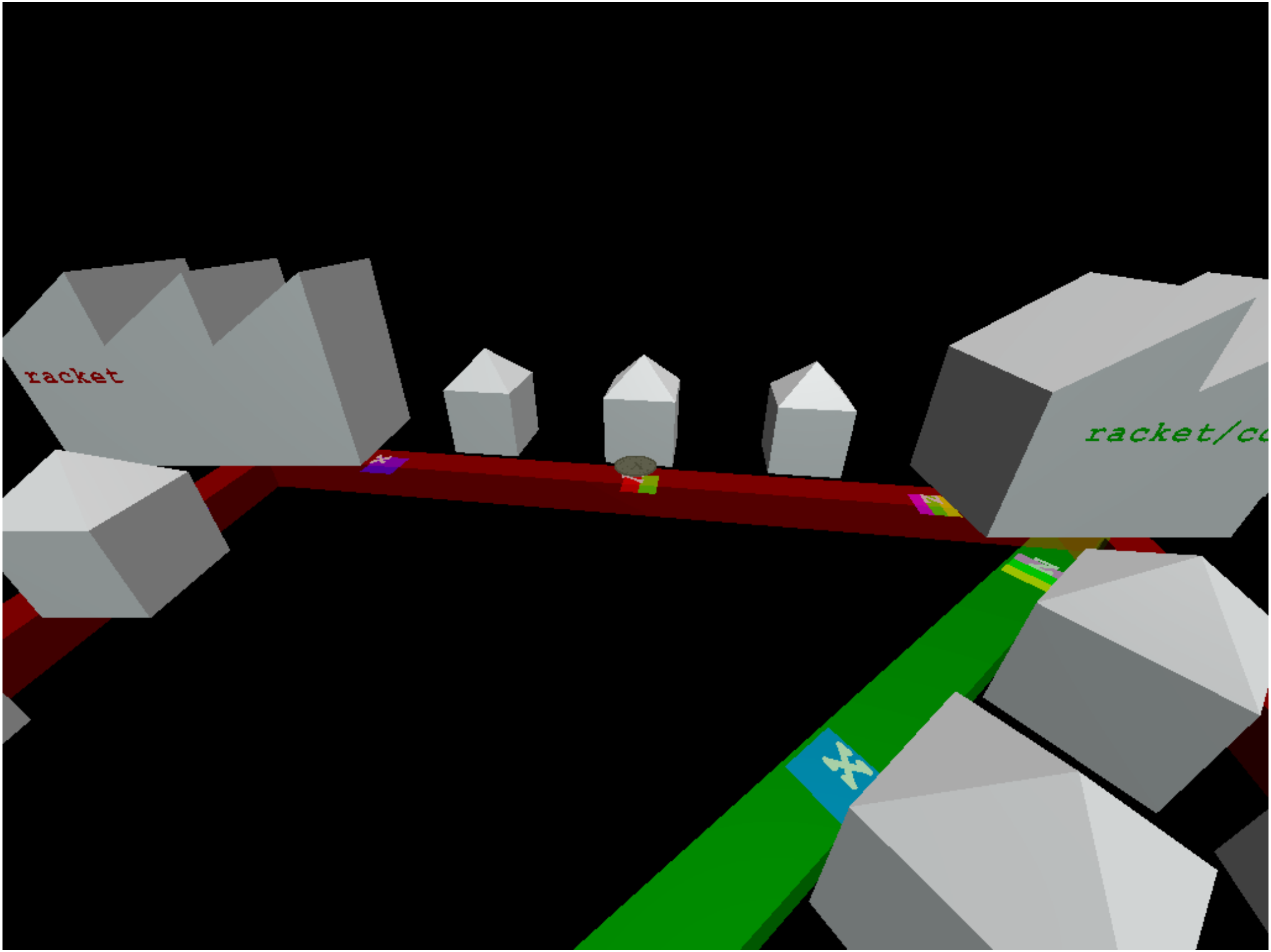


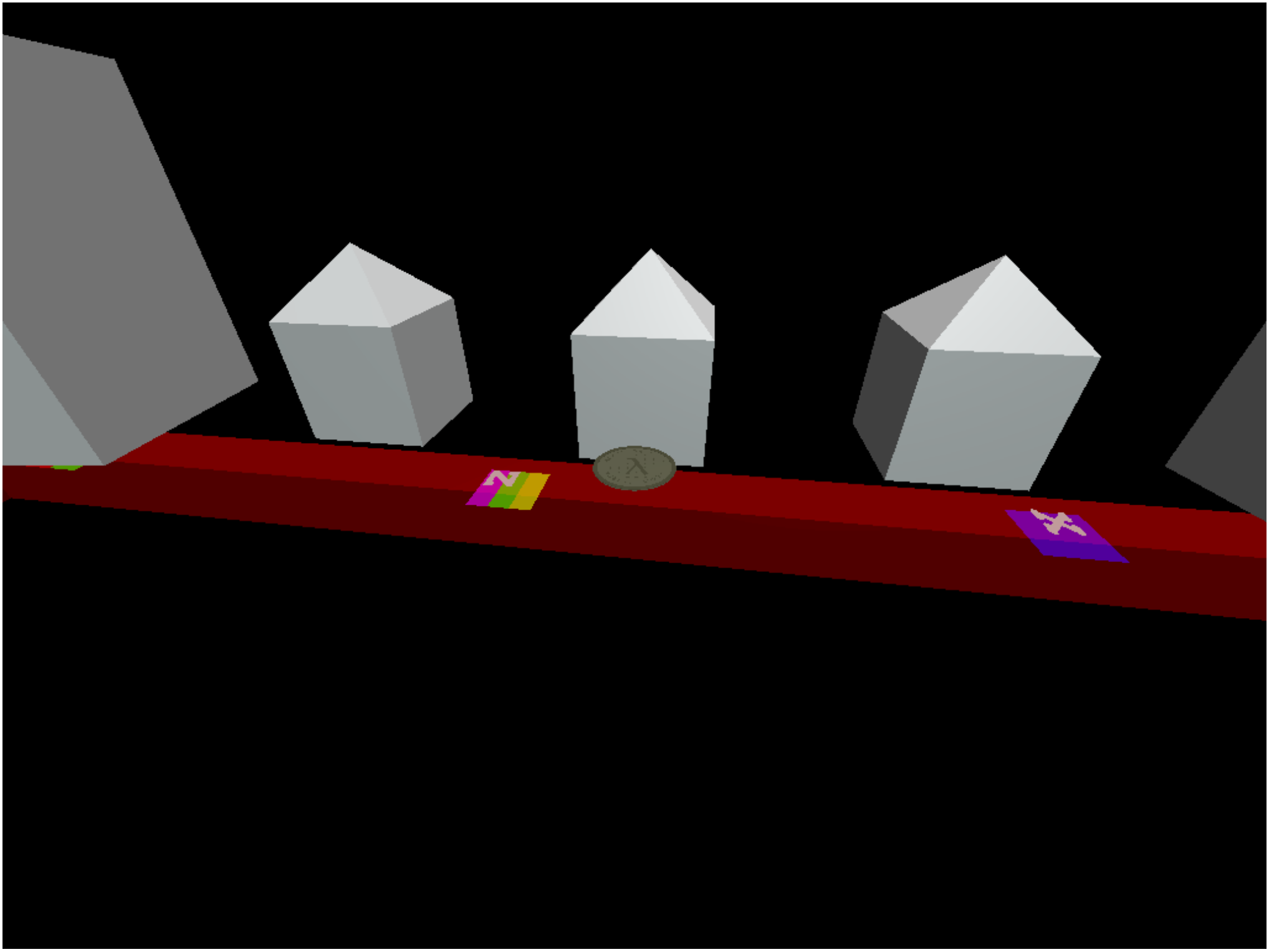


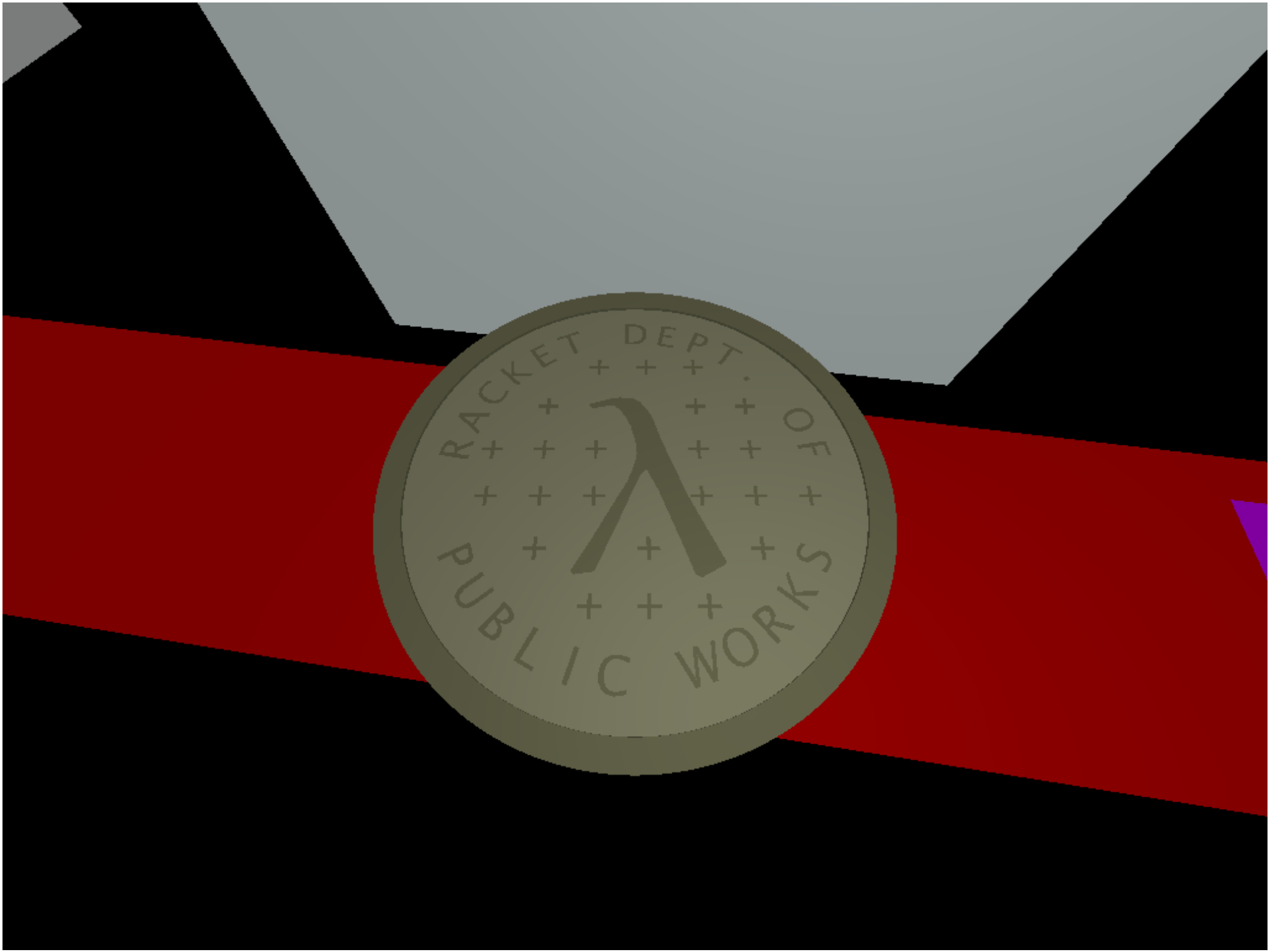





racket
racket/contract


racket
racket/co




RACKET DEPT. OF
PUBLIC WORKS