〔非公開〕

ＴＲ－Ｍ－００３６

軌跡描画による3次元空間のナビゲーション

五十嵐　健夫　　門林　理恵子
Takeo IGARASHI　　Rieko KADOBAYASHI

１９９８．４．１０

ＡＴＲ知能映像通信研究所

軌跡描画による3次元空間のナビゲーション

五十嵐 健夫[1]　　門林 理恵子[2]

[1]東京大学大学院 情報工学専攻　[2](株)ATR 知能映像通信研究所

概要

仮想 3 次元空間のナビゲーション手法として軌跡描画によるナビゲーションを提案する。これは、ユーザが希望する移動経路を画面上に直接自由ストロークとして描き、その上を辿るようにして視点が 3 次元空間内を移動するというものである。従来の方向転換と前進ボタンを組み合わせた移動に比べて、継続的に制御しつづけなくてはならないことからくる負担感を軽減できる他、目的地をマウスのようなポインティングデバイスで選択して直接ジャンプする方式に比べても、少ない手間で目的地とそこへいたるまでの経路と方向を同時に指定できるという点でも優れている。本稿では、この軌跡描画によるナビゲーションについて、その動作と特徴の説明を行い、デモンストレーションシステムと評価実験について述べる。実験の結果、軌跡描画によって作業時間の短縮は確認できなかったものの、ユーザの満足度の向上傾向を確認することができた。

1. はじめに

計算機処理能力の拡大とともに3次元表現を利用したアプリケーションが増えてきており、空間内を効率的に移動するインタラクションテクニックの必要性が増大しつつある。しかし、現在主に用いられている手法は基本的に前進と方向転換といった現在地からの移動量を与え続けるといったもので、ユーザへの負担が大きくまた複雑な経路を生成することも困難である。特に低速の計算機を用いている場合、3 次元シーンの描画速度の不足から移動が極端に遅いといった現象が起こるが、このような場合には特に負担感が大きい。近年これに代わるナビゲーション手法として、ユーザが目的とするオブジェクトをポインティングデバイスなどで直接指定し、システムが自動的に移動経路を計算して移動を行う手法が提案されている [1] が、対象となるオブジェクトのない場合(建物の間を通り抜ける、

など)に利用が困難である他、目的地での方向やそこに至るまでの経路を指定できない、といった限界がある。
本稿では、新しいナビゲーションテクニックとして画面上に直接意図する移動経路を描画する手法を紹介する。本手法により簡単な操作で複雑な移動を効率良く行うことが可能となる。なお、本研究は文献 [3] に発表されている内容をもとに、実装と評価を行ったものである。次章でそのインタラクション手法について解説した後、実際に開発したプロトタイプについて述べ、ついで有効性を確認するために行った評価実験とその結果について説明する。最後に、この手法に関する議論と今後の課題について記し、結論を述べる。

2. 軌跡描画によるナビゲーション

本手法は、ペンやマウスといった 2 次元ポインティングデバイスを用いて 3 次元空間中における視点移動を行うための手法である。ユーザは画面に表示された 3 次元空間の画像の上に目的とする移動経路をフリーハンドで描き、その経路をたどるように視点が移動する(図 1)。内部的には、描かれた自由曲線が仮想空間中の地面の上に投影され、その地面上の軌跡にもとづいて移動経路が生成される(図 2)。軌跡はいつでも描くことができ、描きおわった(マウスボタンが離れたとき)時点で経路を計算し移動を開始する。移動の最中に描画を開始する(マウスボタンが押される)と移動を中止し描画終了をまって新しい移動を開始する。遠方での経路描画は誤差が大きくなりがちなので、このように対話的に経路を修正できる機能は重要である。

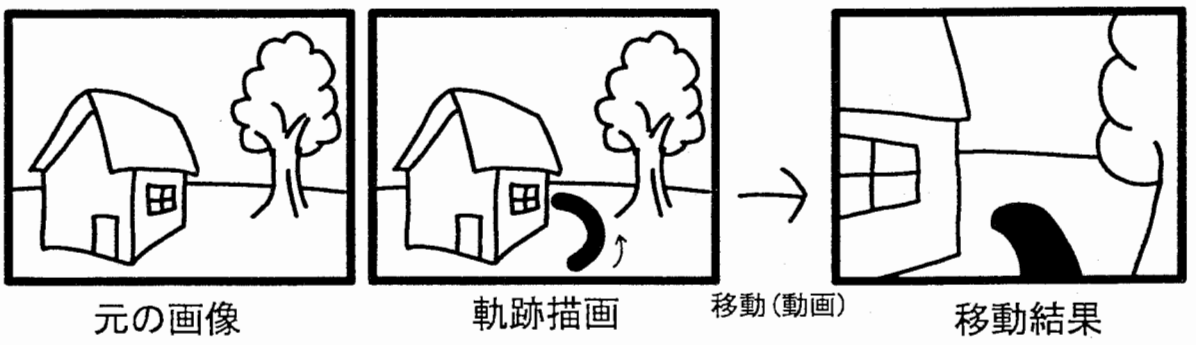

図 1: 軌跡描画によるナビゲーション

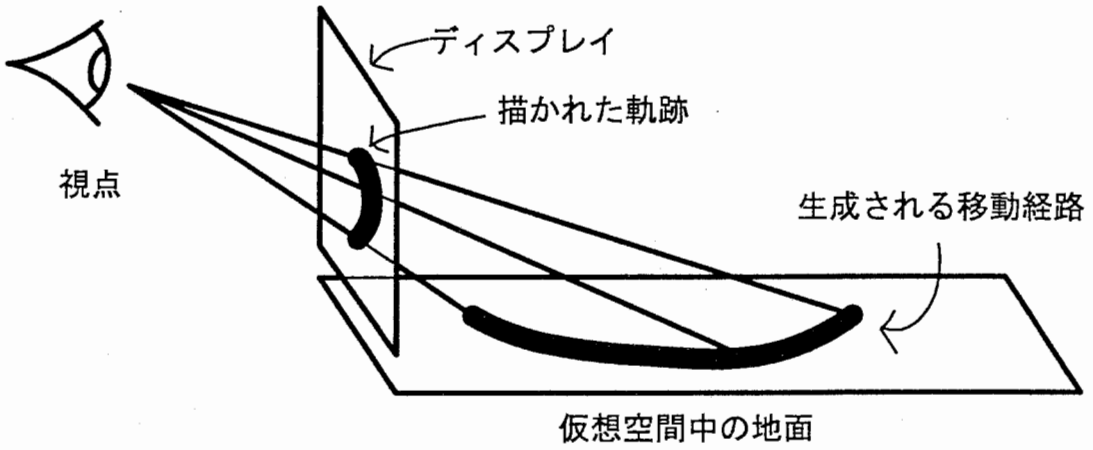

図 2：投影による経路計算

入力から実際の移動経路を求める際、ユーザの描いた軌跡をそのまま利用することも可能であるが、同時に仮想空間の構造にもとづいて障害物を自動的によけるといった処理を加えることにより、より効率的で自然な移動を実現することができる(図 3)。また、仮想空間中に存在するポリゴンの接続情報を利用することにより、坂や階段を上ったり、ゲートをくぐったりといったことも可能である。

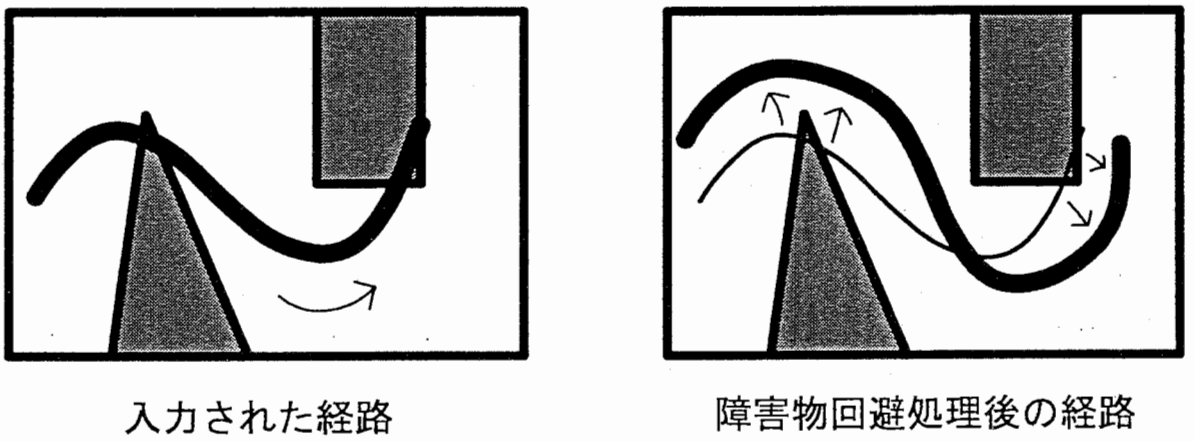

図 3：障害物の自動回避

特に移動経路を指定する必要がなく目的地だけが明らかな場合には、図 4 に示すように短い軌跡を描くことで目的地の位置およびそこでの方向の指示が実現される。このようにすることで、目的地をクリックしてそこへ移動する手法に比べて、ほとんど変わらない手間で位置と方向を同時に指定できる。目的の場所へたどり着くまでの経路は、現在地および目的地とその方向により自動生成可能である。移動を行わずその場で方向転換をしたい場合には、画面の最下部(仮想空間中で足元に相当)で希望する方向に短い線分を描けば良い。

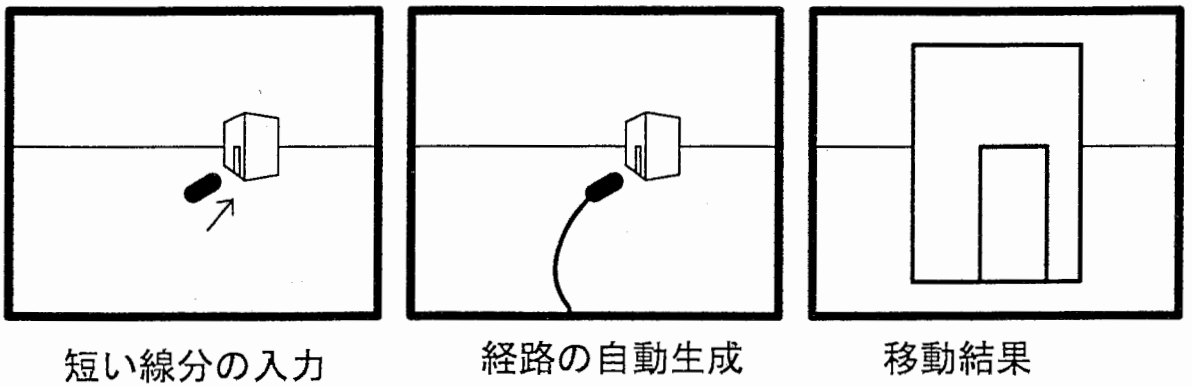

図 4 : 短い軌跡による目的地と方向の同時入力

3. プロトタイプシステムの実装

本手法の有効性や実現性を確認するために Inventor 2.1 を利用したプロトタイプシステムを SGI マシン上に実装した。実装されたシステムは、障害物回避等を含めたデモンストレーション用のものと、他の手法との比較用に作成された実験用のものがあるが、本章では前者について解説する。

3.1. ハードウェア

開発には、バックエンドとして SGI の Octane を使用し、フロントエンドとして Gateway 2000 の PC を利用した。 その際 PC 上で X Server を起動し、そこから Octane に、リモートログインしてウィンドウを開いて利用する。さらにペンタブレットが PC に接続されており、マウスの代わりにディスプレイ一体型ペンタブレットによる入力を使用したテストも行った(図 5)。タブレットによる入力はマウスの動作をエミュレートしているため、コード自体はまったく変更の必要はない (ただし実際に実験等で利用する場合にはそれぞれの特性に応じた調整が必要である)。

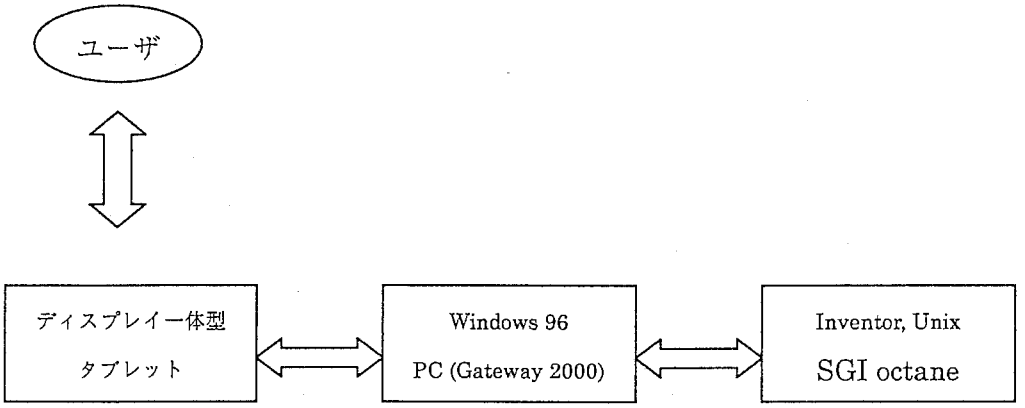

図 5 : ハードウェア構成図

3.2. プログラムの概要

実装されたシステムは基本的には、通常の VRML ブラウザと同様の 3 次元世界のビューワであり、透視投影された 3 次元世界のモデルおよびユーザの分身であるアバターが表示される。そこへユーザがマウスおよびペンで軌跡を描くと、そのストロークが 3 次元世界の歩行面へ投影されて移動がおこる(図 6)。画面上のストロークから仮想空間中への投影は、Inventor の機能である Picking を利用している。ストロークが描かれ、経路が確定した時点で、タイマーイベントが有効になり、一定時間間隔でストローク上を移動していく。スムーズな移動を実現するために、本実装ではポリゴンをそのままなぞるのでなく、入力ストロークの点列を制御点としたBスプラインを経路としている。

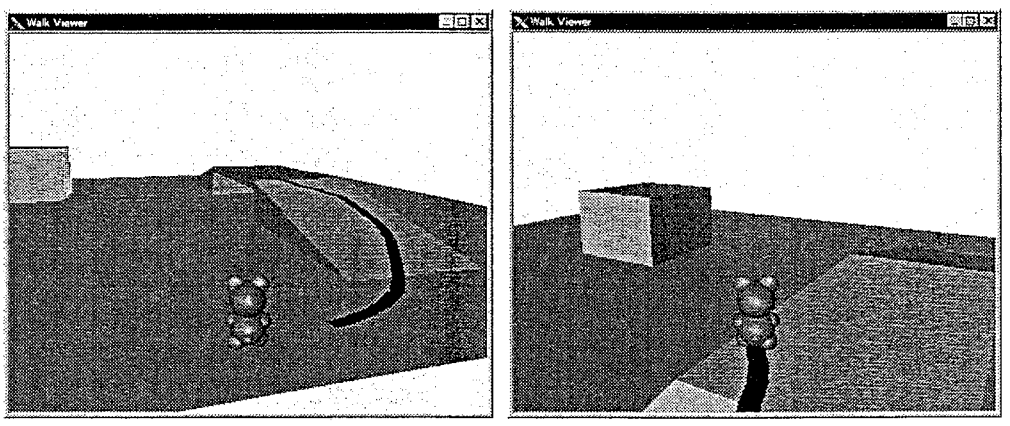

図 6: プロトタイプシステムでの移動例

3.3. 障害物の回避と昇降・通り抜け

手書きストロークが障害物にぶつかっている場合には、その障害物をよけるようにパスが修正される。また、ストローク中の点がゲートの上にあった場合には、直前の点がやはりゲートの上にあればそのまま移動し、それがゲート以外の地面にあった場合には、ゲートの上を通らずにその下を通る。なお、この障害物回避とゲートの昇降・通り抜けのアルゴリズムは、デモンストレーション用に直接コード化したもので一般の仮想空間にすぐに使用することはできない。

VRML で記述されたような一般の仮想空間について、自然な動作を自動的に生成する機構については将来の課題である。

4. 実験

軌跡描画によるナビゲーションの有効性およびその特徴を明らかにすることを目的として、キーボードによる移動と、目的地クリックによる移動を含む実験システムを構成し、被験者の協力のもとで実験を行った。

4.1. システム

ハードウェア構成は前章のプロトタイプのものと同様であり、ディスプレイ一体型タブレットを使用して操作する。ソフトウェアは基本的に同一であるが、簡単のために障害物の自動回避や傾斜の上り下りなどは利用しない。また、実験用の移動用コースを仮想空間中に用意し、障害物にぶつかった場合には移動が停止したり、スタートとゴールの判定をおこなったり、自動的に各種の記録を取る機能などを付加した。

4.2. 被験者

ATR 知能映像通信研究所の研究員 12 人が被験者として参加した。女性 2 人を含む。全員マウスとキーボードによる計算機の使用には習熟しているものの、3 次元空間中の移動に関しては慣れている被験者とそうでない被験者がいた。

4.3. 方法

被験者は、以下にのべる 6 つの条件の元で、仮想空間中のコースをスタートからゴールまで移動する。途中、障害物にぶつかるとそれ以上進めなくなるので、回避しながら進んでいく必要がある。スタートからゴールまでは基本的に一本道であり、またゴールの方向は常に矢印で表示されている。

実験条件:

(1) drive (fast): キーボードによる移動。上下矢印で前進後退、左右矢印で左右回転を行う。上矢印と左右矢印を同時に押すことで回転しながらの前進も可能である。画面の書き換え (移動動作)は 0.1 秒毎に発生する。これは、早すぎても遅すぎても操作しづらい中で、もっとも操作しやすいと思われる設定にしたものである。

(2) drive (slow): キーボードによる移動。(1)と基本的に同一の条件であるが、画面の書き換え速度が半分に設定されている。これは、低速の計算機で描画速度が極端に遅い場合を想定したものである。実際にはこれ以上遅い場合も多いが、実験の都合上 0.2 秒に一回とした。

(3) flying (animation): 目的地をペンでタップするとそこへ移動する。アバターは、現在地と目的地を結んだ直線上を移動し、その速度は drive (fast) と同一である。方向は、一定の角速度で、現在地と目的地を結んだ方向へと変化する。

(4) flying (no animation): (3) と基本的に同じであるが、移動中のアニメーション表示はなく、タップされた瞬間に目的地へジャンプする。ただし、現在地と目的地を結んだ直線が障害物と交わった場合には、その交点で停止する。

(5) drawing (animation): 希望する移動経路を直接画面上に描く。アバターは、現在地からストロークの始点へ直線的に移動したあと、歩行面へ投影されたストロークの上を移動していく。移動速度は (1) と同じであり、方向は移動経路の接線方向で向くように変化する。移動中、障害物にぶつかった場合にはそこで停止する。

(6) drawing (no animation): (5) と同様であるが、移動はアニメーション表示されず、ストロークの描画が終わった時点で、ストロークの終点へジャンプする。この場合も、途中で障害物にぶつかった場合には、そのぶつかった点でジャンプが停止する。

図 7 に実験に使用したプログラムの実行画面例を示す。各被験者は、6 つの条件のもとで順番に実験を行った。一つの条件につき、まずチュートリアルとして一度ゴールまで移動し、その後、スタートからゴールまでの移動を 3 回ずつ行った。6 つの条件の順番は、すべての被験者の間でバランスが取れるように適当に組み替えられている。

図 7: 実験に使用したプログラムの画面例

4.4 結果

図 8 に、ゴールまでにかかった時間の被験者間平均を示す。各被験者については 3 回の試行のうち、もっとも時間の短かったものをとっている。もっとも短い時間でゴールへ到達しているのは、タップによる移動とストロークによる移動のアニメーションのないものであり、次に早いのがキーボードによる移動のうち移動速度が速いものであった。ついで移動中のアニメーションつきのタップによる移動とストロークによる移動がつづき、もっとも遅かったのは、移動速度の遅いキーボードによる移動であった。

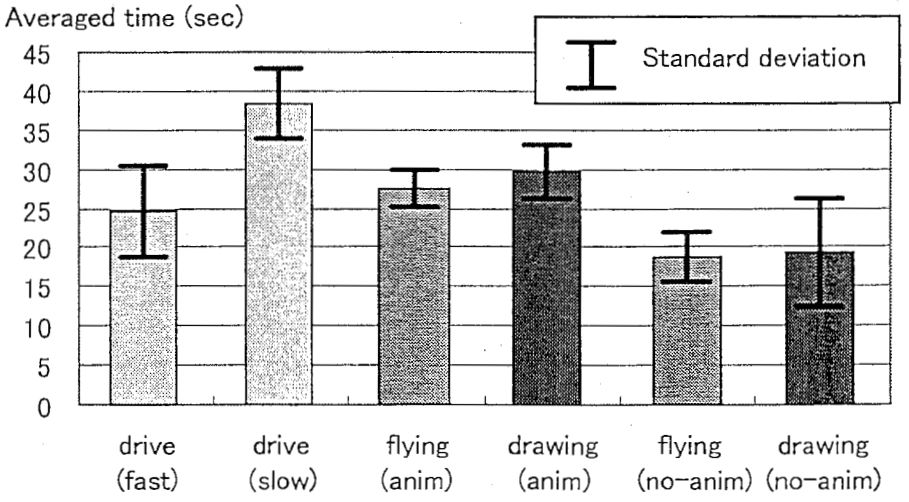

図 8: ゴールへの到達時間の平均と分散

図 9 に実験に関する被験者の主観的評価をアンケート形式で調べた結果を示す。まず「移動時間の長短に関わりなく、仮想空間を歩き回る方法としてどれが気に入ったか」という問い(図 9 上)に対しては、アニメーション付きのタップによる移動とストロークによる移動が高いスコアを得ている。次に高いのが、移動速度の速いキーボードによる移動であり、その後にアニメーションなしのストロークによるもの、移動速度の遅いキーボードによる移動がつづき、もっともスコアの低いのがアニメーションなしのタップによる移動であった。

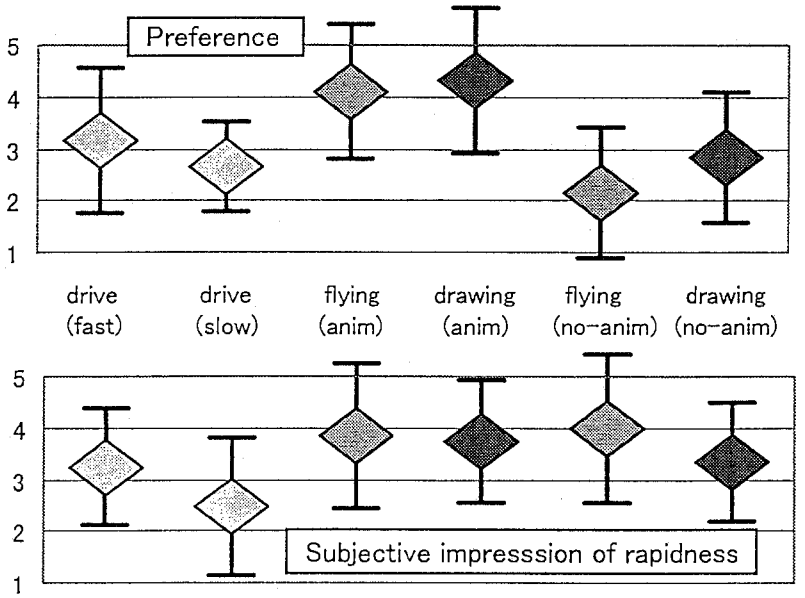

図 9: 実験に関する主観的評価 (5 段階で相対評価をつけるよう指示)
上: 「臨場感」仮想空間を歩き回る方法としてどれが気にいったか
下: 「実感速度」ゴールまでの移動時間がもっとも短く感じられたのはどれか

図 9 下 は、「どの移動方法がもっとも早く感じられたか」という問いに対する答えであり、アニメーションなしのタップ移動が最高得点を得ている。ただ、その後につづくアニメーションありのタップおよびストローク移動とほとんどスコアの差は認められない。その後に、アニメーションなしのストロークによる移動が続き、最後にキーボードによる移動の速いものと遅いものが続いている。

図 10 に、スタートからゴールへいたるまでに、何度シーンの書き換えが行われたかを計測した結果を示す。明らかに移動に伴うアニメーションを生成するキーボードによる移動とアニメーション付きのタップとストローク移動で描画回数が多く、アニメーションを伴わない移動の場合に描画回数が極端にすくないことが分かる。とくにアニメーションなしのストローク移動では、非常に少ない描画回数でゴールへの到達が実現されている。

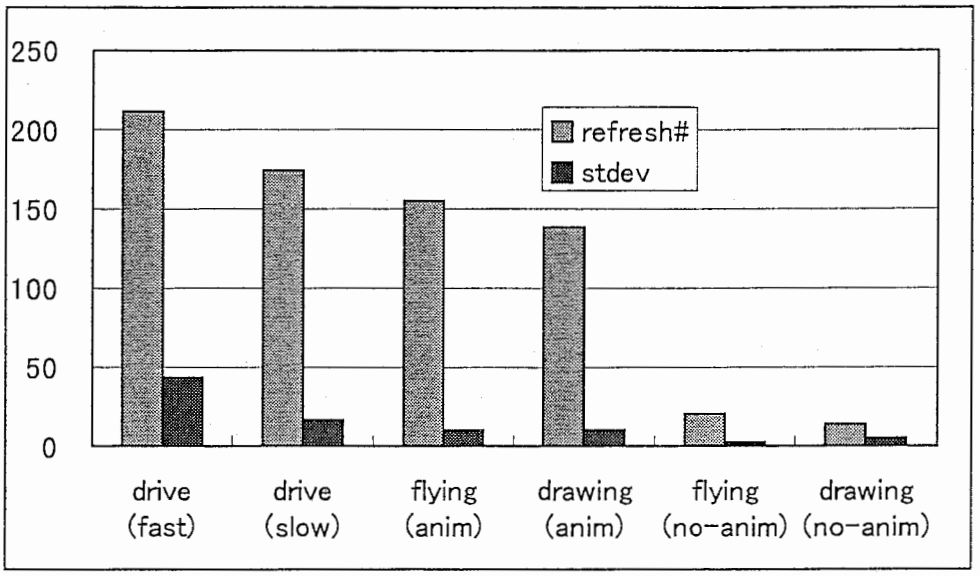

図 10: ゴールにいたるまでの間に起こった画面書き換えの回数

5.5.実験に関する考察

基本的にタップによる移動とストロークによる移動は、キーボードによる移動と比較して似たような結果を示しているといえる。ただ、計測された移動速度に関しては、タップによる移動の方が若干早く、主観的な好みに関してはストロークによる移動の方が多少点が高いという結果になっている。印象としての移動速度に関しては、アニメーションなしのストロークによる移動が特に悪いスコアになっている点が目だつ。これは、アニメーションなしのストロークによる移動において、道に迷う被験者が多かったことに由来すると考えられる。

キーボードによる移動とアニメーション付きのタップおよびストロークによる移動を比較した場合には、実際の計測結果では、キーボードによるものの方が早いにも関わらず、後者二条件の方が主観的評価ではより早く感じられているところが興味深い。主観的な好みでも、ペンを利用した後者二条件がキーボードによる移動にくらべて高いスコアを得ている。

アニメーション無しのタップおよびストロークによる移動と、移動速度を遅く設定したキーボード移動は、計算機の処理能力不足のため画面表示速度が非常に遅い場合を想定したものである。この実験では、ペンを利用した方法によって移動時間がほぼ半分に改善され

主観的な早さでもペンによるもののほうが明らかに高いスコアを得ている。主観的好みに関してはあまり差がないものの、実際のアプリケーションにおいてはこれ以上に条件が悪い (画面の書き換え速度が遅い) 場合も多いと考えられる。図 10 によれば、ペンやマウスを利用した直接的な移動によって描画回数が 10 分の 1 程度になっており、そのような場合に非常に有効な手段を考えられる。

なお、今回の実験に関して、アニメーションなしでのストロークによる移動において、数人の被験者が「道に迷う」現象が観察された。これは、ペンによる入力の際に、ペン先のゆれがストロークに反映されて、被験者の意図しない方向を向いてしまったことなどが原因と考えられる。軌跡描画による移動を実際のペンタブレットで利用する際には、このような点への配慮が必要であるといえよう。

6. 議論

本手法は、マウスでの使用も可能であるが、今後普及が期待されるペンベースのインタフェースとの親和性が高い。また、指先をペンとすることでデータグローブを使用した VR 空間での使用も考えられる。視点から指先(ポインティングデバイス)へ伸びる視線と、その延長上にあるオブジェクトとのインタラクションを実現している点で、本手法は[1]と関連が深い。

ナビゲーションに関しては、目的地さえ指定できれば十分であり経路を指定することの意味はないという意見もあるが、障害物が多く目的地への経路が自明でない場合や、視野外の目的地を探している場合などは、視点の移動を直接指定しながら探索を行なう方が仮の目的地を指定してジャンプするよりも自然であろう。また、目的地をクリック等によって指定する方法ではその点での方向の指定ができない([2]では、対象面に垂直な方向を向くように自動的に設定されるが目的となるオブジェクトがない場合には無効)のに対し、本手法では目的地の指定の際にポインタを少し動かして短い線分を描くだけで方向も同時に指定できる(図 4 参照)点で優れているといえる。

本手法の欠点として、基本的に 2 次元とみることのできる地面上の移動以外の移動動作、すなわち空間中の上昇下降を伴う 3D 空間内での移動が不可能である点が挙げられる。ただし、実世界での人間の移動は基本的に地面の上の移動であり、多くのアプリケーションはこのタイプの移動があれば十分であると考えられる。多少の起伏や階段等に対してはこの手法を直接応用することが可能である。

前進や方向転換といったプリミティブな指示でなく、経路という(計算機の立場からは)よ

り間接的(ユーザからみれば直接的)な方法をもちいることにより、内部処理によって障害物を自動的によけたり移動経路に特殊効果(よっぱらいの歩行など)を与えたりすることが可能となる。また、実世界を移動するロボットからのカメラ映像の上に軌跡を描くことにより遠隔操作に利用することも考えられる。

自由ストロークやクリックによって指定した点列によって、オブジェクトやカメラの移動経路を指定する方法は、市販の 3 次元アニメーション開発システムなどにも見られる。本手法の独自性は、その自由ストロークによる軌跡指定を、デザイン段階で利用するのでなく、対話的な移動方法として利用した点にあるといえる。

7. まとめ

3 次元空間内のナビゲーション手法として画面上に軌跡を直接描画する手法を提案した。本手法により、直感的な操作で複雑な移動経路を生成することが可能になる他、経路生成の過程でさまざまな処理を組み入れることが可能となる。本手法は、方向転換と前進後退を組み合わせた手法に比べて、継続的に指示をあたえ続けなくてはならないという負担を解消している点ですぐれているといえる。また、目的地をクリックしてジャンプする方法と比較しても、目的地の位置のみでなくそこへいたる経路や向きを簡単な操作で一度に指定できるといった利点がある。

実験の結果から、実際の移動時間は本手法によるものとクリックによる移動とでほとんど差がないものの、主観的な好みに関してはクリックによるものよりも高いスコアを得ていることが確認された。特に画面の書き換え速度が遅く、前進ボタンによる移動ではストレスが非常に高いと予想される場面において、軌跡描画による移動が有効であると期待される。

今後は、描かれた軌跡が障害物にぶつかった場合の経路生成アルゴリズムの一般化や実ロボットの遠隔操作、車椅子などの操作といったテーマに取り組んでいく予定である。特に後者の二つに関しては、実世界のビデオ映像の上に直接ストロークを描いて移動するというものを計画しており、従来の移動手段を大きく改善できる可能性がある。

参考文献

1. Mackinlay, J., Card, S.K., Robertson, C.G., Rapid Controlled Movement Through a Virtual 3D Workspace. In *Proc. of SIGGRAPH'90*, pp.171-176, 1990.

2. Pierce, J., Forsberg, A., Conway, M., Hong, S., Zeleznik, R., Image Plane Interaction Techniques in 3D Immersive Environments. In *Proc. of Interactive 3D Graphics'97,* 1997.
3. 五十嵐 健夫、原田　康徳、 尾内 理紀夫、軌跡描画による3次元空間のナビゲーション、「軌跡描画による3次元空間のナビゲーション」 インタラクティブシステムとソフトウェアに関するワークショップ V (WISS'97), pp.119-122, 1997年12月.

番号 _____

お名前 _____

実験のアンケート

ご協力どうもありがとうございました。
簡単なアンケートにお答えください。

3次元の仮想空間中の移動にはどのくらい慣れてますか。(ゲームで、あるいは研究で。)

1　2　3　4　5

(5段階: 1. まったく経験なし, 3. 何度か触ったことある, 5. 頻繁に利用する)

全部で6つの条件で実験していただきましたが
この6つの条件について
「臨場感」(移動時間の長短に関わりなく、仮想空間を歩き回る方法としてどれが気に入ったか。)
「実感速度」(ゴールまでの時間が感覚的にもっとも短く感じられたのはどれか。)
の2点について5段階で評価してください。
相対的にもっとも良いものが 5, 悪いものが 1 になるようにしてください。

	臨場感 1 嫌い　5 好き	実感速度 1 遅い　5 早い
1. キーボードでの移動 (早い)	1 2 3 4 5	1 2 3 4 5
2. キーボードでの移動 (遅い)	1 2 3 4 5	1 2 3 4 5
3. クリックしてジャンプ (アニメーションあり)	1 2 3 4 5	1 2 3 4 5
4. クリックしてジャンプ (アニメーションなし)	1 2 3 4 5	1 2 3 4 5
5. 経路を描いて移動 (アニメーションあり)	1 2 3 4 5	1 2 3 4 5
6. 経路を描いて移動 (アニメーションなし)	1 2 3 4 5	1 2 3 4 5

実験に関する感想等ございましたらご自由にお書きください。
(実験の方法について、それぞれの移動方法について、etc.)

どうもありがとうございました。　　五十嵐 健夫

Path Drawing for 3D Walkthrough

Takeo Igarashi[1], *Rieko Kadobayashi*[2], *Kenji Mase*[2], *Hidehiko Tanaka*[1]

[1]Dept. of Info. Engineering, Univ. of Tokyo
7-3-1 Hongo Bunkyo-ku
Tokyo, 113-8656, JAPAN
Tel: +81 3 3812 2111 ext. 7413
E-mail: {takeo, tanaka}@mtl.t.u-tokyo.ac.jp

[2]ATR Media Integration & Communications
Research Laboratories
Seika-cho, Soraku-gun, Kyoto, 619-0288, JAPAN
Tel: +81 774 95 1443
E-mail: {rieko, mase}@mic.atr.co.jp

ABSTRACT

This paper presents an interaction technique for walkthrough in virtual 3D spaces, where the user draws the intended path directly on the scene, and the avatar automatically moves along the path. The system calculates the path by projecting the stroke drawn on the screen to the walking surface in the 3D world. Using this technique, the user can specify not only the goal position, but also the route to take and the camera direction at the goal with a single stroke. A prototype system is tested using a display-integrated tablet, and experimental results suggest that the technique can enhance existing walkthrough techniques.

KEYWORDS: interaction techniques, virtual spaces, 3D walkthrough, pen computing, user study.

INTRODUCTION

Efficient 3D navigation techniques are required to meet the increasing popularity of virtual spaces. Existing walkthrough techniques can be divided into roughly two categories. One is *driving*, where the user continuously changes the camera position using advancing and turning buttons (arrow keys, joysticks, or button widgets on the screen). The other is *flying*, where the camera automatically jumps to the goal position that the user had specified using a pointing device [1]. Driving is commonly used for computer games, but can cause unwanted overhead when the walking is not the primary purpose of the interaction, because the user has to continuously press buttons during the movement. This problem gets serious especially when the rendering speed is slow, which is often the case with current desktop VR on PCs. Flying provides a solution to the problem, freeing the user from continuous control. All the user has to do is to click the target, then he can arrive at the target position instantly. However, flying suffers from its limited expressive power. The user cannot specify which route to take during the movement, nor can he control the orientation of the camera directly.

PATH DRAWING FOR 3D WALKTHROUGH

We propose a *path drawing* technique for 3D space navigation, which is an enhancement of the flying technique. It allows the user to draw the desired walkthrough path directly on the screen using a free stroke. Then, the system automatically calculates the moving path in the 3D world by projecting the stroke onto the walking surface, and presents the movement of the avatar and the camera in an animated manner. The avatar's direction is fixed to the tangent of the projected stroke. The user can draw a new stroke during the movement to modify the path, which is important because the far end of a stroke can easily get out of control. Figure 1 illustrates an example of path drawing navigation.

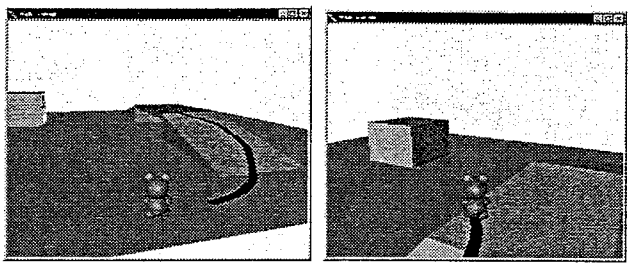

Figure 1: An example of path drawing walkthrough. The user draws the desired path directly on the screen (left), and the avatar and camera move along the projected path (right).

The user can draw either a long stroke specifying the detailed intermediate route to follow, or a very short stroke near the goal position. Long strokes are useful when the user is interested in how to get to the target position, while the user can conveniently specify the goal position and camera direction one at a time using short strokes. The user can also *turn* at the current position by drawing a short stroke at his foot in the intended direction.

This technique can work more effectively when the system is given a detailed structure of the virtual space. For example, it is possible to achieve the automatic avoidance of obstacles when the direct projection of the user's stroke intersects the obstacles. Climbing slopes and going through a gate can be detected by checking the polygon connectivity along the projected path (Figure 1).

A prototype system is developed using Inventor 2.1 on SGI graphics workstations. Automatic obstacle avoidance, slope climbing, and gate through are implemented and tested. However, these additional functions are turned off during the following evaluation.

EVALUATION

An experiment is performed to clarify the characteristics of path drawing against driving and flying techniques. Twelve subjects (computer science researchers) are instructed to get to the specified goal as rapidly as possible, navigating through a virtual space while avoiding obstacles. The subjects perform the task under the following six conditions, in a balanced order. In each condition, the camera is fixed just behind the avatar, and the avatar stops when it collides with an obstacle while traversing. A standard keyboard is used for "driving", while a display integrated tablet is used for "flying" and "drawing".

1) Driving (fast): the user controls the avatar using arrow keys. The left and right keys correspond to turn operations. Each movement occurs every 0.1 sec. (assumed to be the best setting).
2) Driving (slow): The same condition as 1), except that the screen refresh rate (and also the moving speed) is half one-half.
3) Flying (animated): the user clicks the intended position directly, and the avatar smoothly moves toward the target in an animated manner. The moving speed is identical to that in 1).
4) Drawing (animated): the avatar moves along the drawn path in an animated manner. The speed is identical to that in 1).
5) Flying (no animation): flying without animation. The avatar instantly jumps to the target position after a click.
6) Drawing (no animation): path drawing navigation without animation. The avatar's position and direction change instantly to the final state.

Figure 2 shows the averaged elapsed time to get to the goal measured by the system, and Figure 3 shows users' preferences and subjective impressions on how fast they finished the task under each condition. Although drawing and flying exhibit similar results, the latter is a little bit faster when the user slightly prefers path drawing to flying. Another observation is that the users seem to feel that animated flying and drawing faster than driving, while driving (fast) is actually faster than the both of them. Driving (slow) simulates the current tedious 3D navigation when the rendering speed is extremely low; flying and drawing without animation are tested as alternatives in the case. Actual desktop VR can be much slower and more uncomfortable, and these experimental results suggest that flying and path drawing without animation can mitigate the problem.

DISCUSSION

Path drawing can be used with any pointing device, but is most suitable for a pen-based or touch panel system. It is also possible to use this technique in immersive VR environments with HMD and data gloves, where the user draws the path using the finger [2].

A limitation of path drawing is that it cannot be directly applied to completely free 3D movements (not constrained

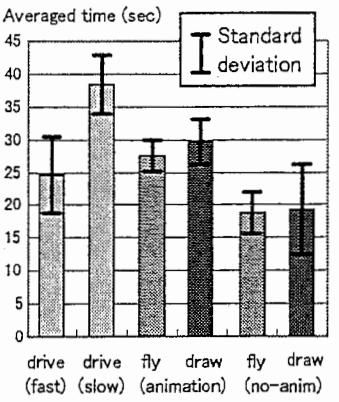

Figure 2 (left): Averaged time to get to the goal. A subject performed six tasks three times each, and the best among the three was selected and averaged.

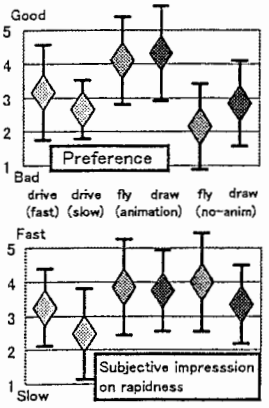

Figure 3 (right): Subjective evaluations. The subjects were required to give relative scores ranging from 1 to 5 depending on their preferences of the techniques and their subjective impressions on rapidness.

to a walking surface). Another limitation is that the avatar must be present on the screen in order for a path to be drawn at the avatar's feet, but this problem may not be so serious because path drawing can naturally coexist with flying and driving in real applications.

CONCLUSIONS AND FUTURE WORK

We presented a technique for 3D virtual space walkthrough, in which the user specifies the intended path by drawing a free stroke on a virtual walking surface on the screen. This technique is superior to conventional *driving* in that the user does not have to continuously control the movement, and extends *flying* by letting the user specify the route and direction one at a time. Experimental results suggest that the technique can achieve more comfortable interaction than others, maintaining a comparable operation speed.

Path drawing navigation is useful especially when the rendering speed is low or the communication delay is large, because the user can give detailed instructions to the computer one at a time using a free stroke, and the computer can take time to perform time-consuming calculations. We plan to apply this technique to remote robot control and wheelchair navigation, where the user draws strokes on real-world video images.

ACKNOWLEDGMENTS

We thank ATR MIC members who helped us in the experiment, and gave us valuable comments.

REFERENCES

1. Mackinlay, J., Card, S.K., Robertson, C.G., Rapid Controlled Movement Through a Virtual 3D Workspace. In *Proc. of SIGGRAPH'90*, pp. 171-176, 1990.

2. Pierce, J., Forsberg, A., Conway, M., Hong, S., Zeleznik, R., Image Plane Interaction Techniques in 3D Immersive Environments. In *Proc. of Interactive 3D Graphics'97*, 1997.

```
// Copyright (c) Takeo Igarashi

// Path Drawing project main program

#include <sys/types.h>
#include <sys/times.h>
#include <stdlib.h>

#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/Text.h>

#include <GL/gl.h>
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtRenderArea.h>
//#include <Inventor/Xt/viewers/SoXtFullViewer.h>
//#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/Xt/viewers/SoXtWalkViewer.h>
#include <Inventor/actions/SoWriteAction.h>
#include <Inventor/actions/SoGetMatrixAction.h>
#include <Inventor/actions/SoSearchAction.h>

#include <Inventor/details/SoFaceDetail.h>

#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoLineSet.h>
#include <Inventor/nodes/SoCoordinate3.h>

#include <Inventor/nodes/SoScale.h>
#include <Inventor/nodes/SoRotation.h>
#include <Inventor/nodes/SoPickStyle.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoDirectionalLight.h>

#include <Inventor/sensors/SoTimerSensor.h>
#include <Inventor/sensors/SoFieldSensor.h>

#include <Inventor/SoPickedPoint.h>
#include <Inventor/actions/SoRayPickAction.h>
#include <Inventor/events/SoMouseButtonEvent.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSwitch.h>


void
print_position(SbVec3f position){
   float x,y,z;
   position.getValue(x,y,z);
   printf("%f, %f, %f \n", x,y,z);
}




SbBool
myButtonPressed (SoNode *root,
   const SbViewportRegion &viewport,
   const SbVec2s &cursorPosition);
void
myMousePressCB(void *userData, SoEventCallback *eventCB);
void
```



```
save(   SoSeparator *root, char *filename);
void
save_binary(   SoSeparator *root, char *filename);
SoSeparator*
load(char *filename);
SoSeparator*
human();
SoSeparator*
line();
void
jump_interimCB(void *data, SoSensor *);
static void
humanChangedCB(void *data, SoSensor *);
static void
cameraChangedCB(void *data, SoSensor *);
SbBool
myAppEventHandler(void *userData, XAnyEvent *anyevent);
void
buttonCB(Widget, void *, void *);
void
buttonCB2(Widget, void *, void *);
void
keyboard_forward(int direction);
void
keyboard_turn_right(int direction);


SoSwitch *humanSwitch;
SoTimerSensor *switchHumanTimer;


SoCamera *camera;
SoSFRotation *cameraRotation;
SoTranslation *selfPosition;
SoRotation *selfRotation;
SoTimerSensor *jumpSensor;

int button;  // for picking
SbBool path_following = FALSE;
SbBool freeze = FALSE;


// experiment
int given_parameter;
int experiment_mode;
const int KEYMOVE = 0;
const int JUMP = 1;
const int PATH = 2;

tms dammy_tms;
clock_t start_time;
clock_t end_time;
float total_length;
float draw_length;
float previous_x;
float previous_y;
clock_t drawing_time;
clock_t drawing_start_time;
//clock_t keypressed_time;
//clock_t keypressed_start_time;
int click_count;
int redraw_count;
int block_count;

#include "utility.cc";
```

```
// demonstration (arch)
#include "park.cc";

// experiment (course)
//#include "course.cc";


/*
// vista
const SbBool vista = TRUE;
const char* human_model_left = "../vista/human1.iv";
const char* human_model_right = "../vista/human2.iv";
*/



Course *course;
#include "path.cc";
Path *path;
#include "keymove.cc";
KeyMove *keymove;
#include "jump.cc";
Jump *jump;

void
main(int argc, char **argv)
{

   if (argc == 2)
     given_parameter = atoi(*++argv);
   else
     given_parameter = 0;
   printf("parameter set #%d\n", given_parameter);


   // Initialize Inventor. This returns a main window to use.
   // If unsuccessful, exit.
   Widget myWindow = SoXt::init(argv[0]); // pass the app name
   if (myWindow == NULL) exit(1);

   // Load a scene
   SoSeparator *root = new SoSeparator;
   root->ref();


   /*
   // Add an event callback to catch mouse button presses.
   // The callback is set up later on.
   SoEventCallback *myEventCB = new SoEventCallback;
   root->addChild(myEventCB);
   */

   // Load a Scene Graph
   //SoSeparator *fileContents = load("../vista/simple.iv");
   course = new Course();
   SoSeparator *fileContents = course->root;
   root->addChild(fileContents);

   // Find Camra in the Scene Graph
   SoSeparator *scene = fileContents;
   //(SoSeparator*)SoNode::getByName("scene");
   camera = (SoPerspectiveCamera*)SoNode::getByName("camera");
   cameraRotation = &(camera->orientation);


   scene->addChild(human());
```



```
   path = new Path(scene);
   keymove = new KeyMove();
   jump = new Jump(scene);




   //Create a WalkViewer
   SoXtWalkViewer *myViewer =
          new SoXtWalkViewer(myWindow);
   //myViewer->setDecoration(FALSE);
   myViewer->setSceneGraph(root);
   myViewer->setViewing(FALSE);
   myViewer->setTitle("Walk Viewer");
   myViewer->show();
   myViewer->setCamera(camera);


   // Button
   Arg args[12];
   int n=0;
   XtSetArg(args[n], XmNheight, 24); n++;
   XtSetArg(args[n], XmNwidth, 24); n++;
   XtSetArg(args[n], XmNy, 24); n++;
   Widget parent = myViewer->getAppPushButtonParent();
   Widget  myButton = XmCreatePushButton(parent, "N", args, n);
   myViewer->addAppPushButton(myButton);
   XtAddCallback(myButton, XmNactivateCallback,
        (XtCallbackProc) buttonCB, NULL);

   Widget  myButton2 = XmCreatePushButton(parent, "R", args, n);
   myViewer->addAppPushButton(myButton2);
   XtAddCallback(myButton2, XmNactivateCallback,
        (XtCallbackProc) buttonCB2, NULL);

   // Have render area send events to us instead of the scene
   // graph.  We pass the render area as user data.
   myViewer->setEventCallback(
              myAppEventHandler, myViewer);

   // Walk animation
   jumpSensor =
     new SoTimerSensor(jump_interimCB, camera);
   jumpSensor->setInterval(interval);
   jumpSensor->schedule();

   course->init_parameters();



   // sub window top view
   /*
   SoSeparator *subroot = new SoSeparator;
   //subroot->ref();
   SoPerspectiveCamera *subcamera = new SoPerspectiveCamera();
   subcamera->position.setValue(50, 180, -100);
   subcamera->orientation.setValue(SbVec3f(1,0,0),-1.57);
   subroot->addChild(subcamera);
   subroot->addChild(scene);
   SoXtWalkViewer *subViewer = new SoXtWalkViewer();
   subViewer->setDecoration(FALSE);
   subViewer->setSceneGraph(subroot);
   subViewer->setTitle("map view");
   subViewer->show();
```

```
    subViewer->setCamera(subcamera);
    */


    SoXt::show(myWindow);  // Display main window
    SoXt::mainLoop();      // Main Inventor event loop
}



//  timered callback for jump
void
jump_interimCB(void *data, SoSensor *)
{

  if (freeze)
    return;

  // path following
  if (path_following){
    if (!path->auto_move(&(selfPosition->translation),
                         &(selfRotation->rotation)))
      return;
  }
  else if (keymove->moving()){
    if (!keymove->auto_move())
      return;
  }
  else{
    if (!jump->auto_move(&(selfPosition->translation),
                         &(selfRotation->rotation)))
      return;
  }

  redraw_count++;

  // leg animation
  int index = (int)(humanSwitch->whichChild.getValue());

  if (index == 0)  humanSwitch->whichChild = 1;
  else             humanSwitch->whichChild = 0;
}



// Pick
SbBool
tryPick(SoNode *root,
    const SbViewportRegion &viewport,
    const SbVec2s &cursorPosition,
    SbVec3f *picked_position)
{

    SoRayPickAction myPickAction(viewport);

    // Set an 8-pixel wide region around the pixel
    myPickAction.setPoint(cursorPosition);
    myPickAction.setRadius(8.0);
    myPickAction.setPickAll(TRUE);


    // Start a pick traversal
    myPickAction.apply(root);
    const SoPickedPoint *myPickedPoint =
             myPickAction.getPickedPoint();
```



```
    if (myPickedPoint == NULL) return FALSE;

    SbVec3f position =  myPickedPoint->getPoint();
    position =  course->pickposition(&myPickAction, &position);
    if (position == NULL)
      return FALSE;
    *picked_position = position;

    return TRUE;
}



SbVec3f
foot_position(){
  float x,y,z;
  selfPosition->translation.getValue().getValue(x,y,z);
  return SbVec3f(x,-1,z);
}

// Direct Event Handling
SbBool
myAppEventHandler(void *userData, XAnyEvent *anyevent)
{
    SoXtRenderArea *myRenderArea = (SoXtRenderArea *) userData;
    XButtonEvent *myButtonEvent;
    XMotionEvent *myMotionEvent;
    XKeyEvent *myKeyEvent;
    SbVec3f vec;
    SbBool handled = TRUE;

    const SbViewportRegion &myRegion =
    myRenderArea->getViewportRegion();
    SoGroup *root = (SoGroup *) myRenderArea->getSceneGraph();

    short window_w, window_h;
    myRegion.getWindowSize().getValue(window_w,window_h);

    SbVec3f picked_position;

    switch (anyevent->type) {

    case ButtonPress:
       myButtonEvent = (XButtonEvent *) anyevent;
       click_count++;
       button = myButtonEvent->button;
       if ((myButtonEvent->button == Button1) && (experiment_mode != JUMP)) {
         // path drawing
         drawing_start_time = times(&dammy_tms);
         path->clear();
         const SbVec2s mouse_position
           (myButtonEvent->x, window_h - myButtonEvent->y);
         if (tryPick(root, myRegion, mouse_position, &picked_position)){
           //path->addPoint(foot_position());
           path->addPoint(picked_position);
           jumpSensor->unschedule();
         }
       } else if ((myButtonEvent->button == Button2)
                  || (experiment_mode == JUMP)){
         // jump
         const SbVec2s mouse_position
           (myButtonEvent->x, window_h - myButtonEvent->y);
         if (tryPick(root, myRegion, mouse_position, &picked_position))
           jump->init_move(root, picked_position);
           //start_jump(root, picked_position);
         path_following = FALSE;
```

```
      jumpSensor->unschedule();
    } else if (myButtonEvent->button == Button3) {
    }
    break;

case ButtonRelease:
   myButtonEvent = (XButtonEvent *) anyevent;
   // path
   if ((myButtonEvent->button == Button1) && (experiment_mode != JUMP)) {
     drawing_time += times(&dammy_tms) - drawing_start_time;
     if (animation){
        path_following = TRUE;
        if (!path->init_move(&(selfPosition->translation),
              &(selfRotation->rotation))){
          //single click
          path_following = FALSE;
        }
     }
     else {
       redraw_count++;
       path->jump(&(selfPosition->translation),
                  &(selfRotation->rotation));
     }
     jumpSensor->schedule();
   // jump
   } else if ((myButtonEvent->button == Button2)
              || (experiment_mode == JUMP)){
     jumpSensor->schedule();
     if (!animation){
       redraw_count++;
       jump->jump(&(selfPosition->translation),
                  &(selfRotation->rotation));
     }
   }
   break;

case MotionNotify:
   myMotionEvent = (XMotionEvent *) anyevent;
   if ((myMotionEvent->state & Button1Mask)&& (experiment_mode != JUMP)) {
     const SbVec2s mouse_position
       (myMotionEvent->x, window_h - myMotionEvent->y);
     if (tryPick(root, myRegion, mouse_position, &picked_position))
       path->addPoint(picked_position);
   }
   break;

case KeyPress:
   myKeyEvent = (XKeyEvent *) anyevent;
   keymove->init_move(myKeyEvent->keycode);
   if (myKeyEvent->keycode == 102) {freeze = !freeze;       break;}

   //keypressed_start_time = times(&dammy_tms);
   path_following = FALSE;
   break;
case KeyRelease:
   //keypressed_time += times(&dammy_tms)-keypressed_start_time;
   myKeyEvent = (XKeyEvent *) anyevent;
   keymove->stop_move(myKeyEvent->keycode);
   break;

default:
   handled = FALSE;
   break;
}

return handled;
```



```
}



// Button
void
buttonCB(Widget, void *, void *)
{
   path->clear();
   jump->clear();
  course->set_next_parameters();
}
void
buttonCB2(Widget, void *, void *)
{
   path->clear();
   jump->clear();
  course->reset_parameters();
}



// human shape
void  switchHumanCB(void *, SoSensor *);
SoSeparator*  human()
{
  // Human Root
  SoSeparator* human2 = new SoSeparator;

  // Pick Style
  SoPickStyle *style2 = new SoPickStyle;
  style2->style = SoPickStyle::UNPICKABLE;
  human2->addChild(style2);

  selfPosition = new SoTranslation;
  human2->addChild(selfPosition);
  selfRotation = new SoRotation;
  selfRotation->rotation.setValue(SbVec3f(0,1,0),0.1);
  human2->addChild(selfRotation);

  // add Sensor for the camera to follow human
  SoFieldSensor *mySensor =
          new SoFieldSensor(humanChangedCB, selfPosition);
  mySensor->attach(&selfPosition->translation);

  SoFieldSensor *myDirectionSensor =
          new SoFieldSensor(humanChangedCB, selfPosition);
  myDirectionSensor->attach(&selfRotation->rotation);

  // Transform to original position
  SoTransform* humanTransform2 = new SoTransform;
  humanTransform2->translation.setValue(0.0, 2.0, 0.0);
  human2->addChild(humanTransform2);



  // Human Model (Switch Node)
  SoSeparator *tmpHumanL;
  SoSeparator *tmpHumanR;
  if (vista){
    tmpHumanL = (SoSeparator *)load("../vista/human1.iv");
    tmpHumanR = (SoSeparator *)load("../vista/human2.iv");
```

```
  }
  else{
    //tmpHumanL = (SoSeparator *)load("conel.iv");
    //tmpHumanR = (SoSeparator *)load("cone2.iv");
    tmpHumanL = (SoSeparator *)load(human_model_left);
    tmpHumanR = (SoSeparator *)load(human_model_right);
  }
  humanSwitch = new SoSwitch;
  humanSwitch->addChild(tmpHumanL);
  humanSwitch->addChild(tmpHumanR);

  human2->addChild(humanSwitch);
  humanSwitch->whichChild = 1;

  return human2;
}
// leg animaiton
void  switchHumanCB(void *, SoSensor *)
{
  int index = (int)(humanSwitch->whichChild.getValue());
  if (index == 0)  humanSwitch->whichChild = 1;
  else             humanSwitch->whichChild = 0;
}




// fix camera behind the human
static void
humanChangedCB(void *data, SoSensor *)
{

   /*
   // demonstration ENDING
   float camera_x, camera_y, camera_z;
   camera->position.getValue().getValue(camera_x,camera_y,camera_z);
   camera->position.setValue(camera_x, camera_y + 0.1, camera_z - 0.05);

   camera->orientation = SbRotation(SbVec3f(1,0,0), -camera_y/40) ;
   */

   float human_x,human_y,human_z;
   selfPosition->translation.getValue().getValue(human_x,human_y,human_z);

   float camera_x, camera_y, camera_z;
   camera->position.getValue().getValue(camera_x,camera_y,camera_z);


  // move camera
  SbMatrix      mat0;
  SbVec3f       directionX(1.0, 0.0, 0.0), newDirectionX;
  SbVec3f       directionZ(0.0, 0.0, 1.0), newDirectionZ;

   selfRotation->rotation.getValue().getValue(mat0);
   mat0.multVecMatrix(directionX, newDirectionX);
   mat0.multVecMatrix(directionZ, newDirectionZ);

   float goal_x, goal_y, goal_z;
   goal_x = human_x + newDirectionZ[0] * (camera_distance);
   goal_y = human_y + camera_height;
   goal_z = human_z + newDirectionZ[2] * (camera_distance);

   camera->position.setValue(goal_x, goal_y, goal_z);
   camera->orientation = SbRotation(SbVec3f(1,0,0), -0.2)
```



```
     * selfRotation->rotation.getValue();
}




SoSeparator* load(char *filename){
   // Read the object from a file and add to the scene
   SoInput myInput;
   if (!myInput.openFile(filename) )
      exit (1);
   SoSeparator *fileContents = SoDB::readAll(&myInput);
   if (fileContents == NULL)
      exit (1);
   return fileContents;
}


void save(SoSeparator *root, char *filename){
   SoWriteAction myAction;
   myAction.getOutput()->openFile(filename);
   myAction.getOutput()->setBinary(FALSE);
   myAction.apply(root);
   myAction.getOutput()->closeFile();
}

void save_binary(SoSeparator *root, char *filename){
   SoWriteAction myAction;
   myAction.getOutput()->openFile(filename);
   myAction.getOutput()->setBinary(TRUE);
   myAction.apply(root);
   myAction.getOutput()->closeFile();
}
```

```
// Copyright (c) Takeo Igarashi

// Path Drawing project
//    module for experiment

// common parameters
const SbBool vista = FALSE;
SbBool animation = TRUE;
char* human_model_left = "cone1.iv";
char* human_model_right = "cone2.iv";

float speed = 4;
float rotation_speed = 0.1;
float interval = 0.02;
const float camera_distance = 10;
const float camera_height = 4.2;



#include <fstream.h>

SoRotation *arrowRotation;
SoRotation *arrowRotation2;
SoTranslation *arrowPosition;

// fix arrow to camera
static void
cameraChangedCB(void *data, SoSensor *){
   arrowPosition->translation = camera->position.getValue();
   arrowRotation->rotation = camera->orientation.getValue();

   SbVec3f axis, new_axis;
   float radian;
   selfRotation->rotation.getValue().getValue(axis, radian);
   SbRotation(SbVec3f(1,0,0),-1.57).multVec(axis, new_axis);
   arrowRotation2->rotation = SbRotation(new_axis, radian);

}


class Course{
  public:
    SoSeparator *root;
    SoGroup *circles;

    int circles_num;
    float circles_x[100];
    float circles_y[100];
    float circles_r[100];
    Course(){
      if (vista){
        root = load("../vista/saved.iv"); // simple
        return;
      }

      root = load("course.iv");
      root->ref();
      circles = (SoSeparator *)SoNode::getByName("circles");

      /*
      SoSeparator* circle = new SoSeparator;
      SoTranslation* translation= new SoTranslation;
      translation->translation.setValue(10.0, 0.0, -20.0);
      circle->addChild(translation);
      SoCylinder* cylinder = new SoCylinder;
      cylinder->radius = 2;
```



```
      cylinder->height = 0;
      circle->addChild(cylinder);
      circles->addChild(circle);
      */


      // arrow
      SoSeparator *scene = (SoSeparator*)SoNode::getByName("scene");
      camera = (SoPerspectiveCamera*)SoNode::getByName("camera");
      SoSeparator *arrow = load("arrow.iv");
      arrow->ref();
      arrowPosition = new SoTranslation;
      arrow->insertChild(arrowPosition, 0);
      arrowRotation = new SoRotation;
      arrow->insertChild(arrowRotation, 1);
      arrowRotation2 = (SoRotation*)SoNode::getByName("direction");
      //arrow->insertChild(arrowRotation2, 2);
      scene->addChild(arrow);
      SoFieldSensor *arrowSensor =
        new SoFieldSensor(cameraChangedCB, arrowPosition);
      arrowSensor->attach(&camera->position);




      circles_num = 0;

      fstream s;
      s.open("circles", ios::in);
      float x,y,r;
      while(! s.eof()){
        s >> x;
        s >> y;
        s >> r;
        //printf("%f \n",x,y,r);
        add_circle(x,y,r);
      }
      s.close();
    };

    SoCylinder* current_cylinder;
    SoTranslation* current_translation;
    float center_x;
    float center_y;
    void add_circle(float x, float y, float r){

      SoSeparator* circle = new SoSeparator;
      SoTranslation* translation= new SoTranslation;
      translation->translation.setValue(x, 0, y);
      circle->addChild(translation);

      SoCylinder* cylinder = new SoCylinder;
      cylinder->radius = r;
      cylinder->height = 0;
      circle->addChild(cylinder);

      current_cylinder = cylinder;
      current_translation = translation;
      center_x = x;
      center_y = y;

      circles->addChild(circle);

      circles_x[circles_num] = x;
```

```
    circles_y[circles_num] = y;
    circles_r[circles_num] = r;
    circles_num++;

}

float get_height(float mx, float my, float mh){
  return 0.2;
}

SbBool blocked(float mx, float my){

  if (vista) return FALSE;

  if (start_time == 0){
    start_time =  times(&dammy_tms);
    //printf("start!\n");
  }
  if ((my < -198) && (end_time == 0)){
    end_time =  times(&dammy_tms);
    int elepsed_time = end_time - start_time;
    printf("time, length, draw_l, draw_t, click#, refresh#, block#\n");
    printf("%f, %f, %f, %f, %d, %d, %d\n",
           elepsed_time/100.0,
           total_length,
           draw_length,
           drawing_time/100.0,
           click_count,
           redraw_count,
           block_count
           );

    /*
    printf("elapsed_time %f\n", (end_time - start_time)/100.0);
    printf("total_length %f\n", total_length);
    printf("drawing_time %f\n", drawing_time/100.0);
    //printf("keypressed_time %f\n", keypressed_time/100.0);
    printf("click_count %d\n", click_count);
    printf("redraw_count %f\n", redraw_count);
    */
  }


  if (mx<0 || mx>100 || my>0 || my <-250){
    block_count++;
    return TRUE;
  }

  int num = circles->getNumChildren();
  float x,y,r;
  for (int i=0; i < num; i++){
    get_xyr(circles->getChild(i),&x,&y,&r);
    if (get_distance(x,y, mx,my) < r){
      block_count++;
      return TRUE;
    }
  }

  // OK!!
  total_length += get_distance(mx,my, previous_x, previous_y);
  previous_x = mx;
  previous_y = my;
  return FALSE;
}
```



```
SbVec3f  pickposition(SoRayPickAction *myPickeAction, SbVec3f *position){
  return *position;
}




    void resize_circle(float x, float y){
      current_cylinder->radius =
        get_distance(center_x, center_y, x,y);
      //current_translation->translation.setValue(
      //  (center_x+x)/2,0,(center_y+y)/2);
    }
    void remove_circle(SoSeparator* circle){
      circles->removeChild(circle);
    }
    void get_xyr(SoNode* circle, float *x,float *y,float *r){

      // Find
      SoSearchAction mySearchAction;
      mySearchAction.setType(SoTranslation::getClassTypeId());
      mySearchAction.setInterest(SoSearchAction::FIRST);
      mySearchAction.apply(circle);
      SoTranslation* translation =
        (SoTranslation*) (mySearchAction.getPath()->getTail());
      float dammy;
      translation->translation.getValue().getValue(*x, dammy, *y);

      // Find
      mySearchAction.setType(SoCylinder::getClassTypeId());
      mySearchAction.apply(circle);
      SoCylinder* cylinder =
        (SoCylinder*) (mySearchAction.getPath()->getTail());
      *r = cylinder->radius.getValue();
    }
    void save(int n){

      int num = circles->getNumChildren();

      printf("cn %d\n", num);
      fstream s;
      s.open("circles", ios::out);
      float x,y,r;
      for (int i=0; i < num; i++){
        get_xyr(circles->getChild(i),&x,&y,&r);
        s << x;
        s << ' ';
        s << y;
        s << ' ';
        s << r;
        s << '\n';
      }
      s.close();
    }



int reset_num;
int parameter_num;
void
set_next_parameters();
void
```

```
reset_parameters();
void
init_parameters();

};

// for experiment

// parameters
const float parameters_speed[3] = {2,4,8};
const float parameters_rotation_speed[3] = {0.05,0.1,0.2};
const float parameters_interval[3] = {0.1, 0.2}; //{0.05, 0.1, 0.2};



void
Course::reset_parameters(){
  // times
   start_time = 0;
   end_time = 0;
   drawing_time = 0;
   //keypressed_time = 0;
   total_length = 0;
   draw_length = 0;
   previous_x = 50;
   previous_y = 0;
   click_count = 0;
   redraw_count = 0;
   block_count = 0;

   // start position
   selfPosition->translation = SbVec3f(50,0,0);
   selfRotation->rotation = SbRotation(SbVec3f(0,1,0), 0);


   printf("trial #%d\n", reset_num++);
 }

void
Course::set_next_parameters(){


   parameter_num++;
   if (parameter_num == 6) parameter_num = 0;

   int n = (int)(parameter_num / 3.0);  // 0 or 1
   int m = mod(parameter_num, 3);       // 0,1,2

   // exchange based on given_parameter (balancing)
   int g = mod(given_parameter,6); // 0-5
   if ((int)(g / 3.0 ) == 1)
     n = 1 - n;
   g = mod(g, 3); // 0,1,2
   m = mod((m + g), 3);

   printf("\n");

   if (n == 0){
     animation = TRUE;
     interval = 0.1;
     printf("fast, anim, ");
   }
   else {
     animation = FALSE;
     interval = 0.2;
     printf("slow, no-anim, ");
```



```
   }
   jumpSensor->setInterval(interval);

   if (m == 0) {
     experiment_mode = KEYMOVE;
     printf("keyboard\n");
   }
   else if (m == 1) {
     experiment_mode = JUMP;
     printf("click and jump\n");
   }
   else {
     experiment_mode = PATH;
     printf("path drawing\n");
   }


   reset_num = 0;
   reset_parameters();

}

void
Course::init_parameters(){
  reset_num = 0;
  parameter_num = -1;
  set_next_parameters();
}
```

```
//   printf("hello\n");


#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/Text.h>

#include <GL/gl.h>
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtRenderArea.h>
//#include <Inventor/Xt/viewers/SoXtFullViewer.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
//#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/actions/SoWriteAction.h>
#include <Inventor/actions/SoGetMatrixAction.h>
#include <Inventor/actions/SoSearchAction.h>

#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoLineSet.h>
#include <Inventor/nodes/SoCoordinate3.h>

#include <Inventor/nodes/SoScale.h>
#include <Inventor/nodes/SoRotation.h>
#include <Inventor/nodes/SoPickStyle.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoDirectionalLight.h>

#include <Inventor/sensors/SoTimerSensor.h>
#include <Inventor/sensors/SoFieldSensor.h>

#include <Inventor/SoPickedPoint.h>
#include <Inventor/actions/SoRayPickAction.h>
#include <Inventor/events/SoMouseButtonEvent.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSwitch.h>


void print_position(SbVec3f position){
   float x,y,z;
   position.getValue(x,y,z);
   printf("%f, %f, %f \n", x,y,z);
}



SbBool
myButtonPressed (SoNode *root,
   const SbViewportRegion &viewport,
   const SbVec2s &cursorPosition);
void
myMousePressCB(void *userData, SoEventCallback *eventCB);
void
save(   SoSeparator *root, char *filename);
void
save_binary(   SoSeparator *root, char *filename);
SoSeparator*
load(char *filename);
SoSeparator*
human();
SoSeparator*
mark();
```



```
SoSeparator*
line();
void
jump_interimCB(void *data, SoSensor *);
static void
humanChangedCB(void *data, SoSensor *);
SbBool
myAppEventHandler(void *userData, XAnyEvent *anyevent);
void
buttonCB(Widget, void *, void *);
void
keyboard_forward(int direction);
void
keyboard_turn_right(int direction);


SoCamera *camera;
SoSFRotation *cameraRotation;
SoTranslation *selfPosition;
SoRotation *selfRotation;
SoTranslation *markPosition;
SoTimerSensor *jumpSensor;

float jump_dx = 0;
float jump_dz = 0;
int   jump_step = 0;
float jump_dangle;
SbRotation jump_drotation;

SbBool path_following = FALSE;
const float speed = 1;
const float rotation_speed = 0.2;
const float interval = 0.1;
const float camera_distance = 12;


float Abs(float x){
  if (x>0)
    return x;
  else
    return -x;
}
float Sign(float x){
  if (x>0)
    return 1;
  else
    return -1;
}

#include "path.cc";
#include "course.cc";
Course *course;

void
main(int , char **argv)
{
   // Initialize Inventor. This returns a main window to use.
   // If unsuccessful, exit.
   Widget myWindow = SoXt::init(argv[0]); // pass the app name
   if (myWindow == NULL) exit(1);

   // Load a scene
   SoSeparator *root = new SoSeparator;
   root->ref();
```

```
    /*
    // Add an event callback to catch mouse button presses.
    // The callback is set up later on.
    SoEventCallback *myEventCB = new SoEventCallback;
    root->addChild(myEventCB);
    */

    // Load a Scene Graph
    //SoSeparator *fileContents = load("course.iv");
    course = new Course();
    SoSeparator *fileContents = course->root;
    root->addChild(fileContents);

    // Find Camra in the Scene Graph
    camera = (SoPerspectiveCamera*)SoNode::getByName("camera");

    cameraRotation = &(camera->orientation);


    //Create a ExaminerViewer
    SoXtExaminerViewer *myViewer =
            new SoXtExaminerViewer(myWindow);
    //myViewer->setDecoration(FALSE);
    myViewer->setSceneGraph(root);
    myViewer->setViewing(FALSE);
    myViewer->setTitle("Walk Viewer");
    myViewer->show();
    myViewer->setCamera(camera);


    camera->position.setValue(50, 180, -100);
    camera->orientation.setValue(SbVec3f(1,0,0),-1.57);


    // Have render area send events to us instead of the scene
    // graph.  We pass the render area as user data.
    myViewer->setEventCallback(
                myAppEventHandler, myViewer);

    SoXt::show(myWindow);  // Display main window
    SoXt::mainLoop();      // Main Inventor event loop
}


// Pick
SbBool
tryPick(SoNode *root,
    const SbViewportRegion &viewport,
    const SbVec2s &cursorPosition,
    SbVec3f *picked_position)
{

    SoRayPickAction myPickAction(viewport);

    // Set an 8-pixel wide region around the pixel
    myPickAction.setPoint(cursorPosition);
    myPickAction.setRadius(8.0);

    // Start a pick traversal
    myPickAction.apply(root);
    const SoPickedPoint *myPickedPoint =
            myPickAction.getPickedPoint();
    if (myPickedPoint == NULL) return FALSE;

    SbVec3f position =  myPickedPoint->getPoint();
    *picked_position = position;
```



```
    return TRUE;
}
// Pick
SbBool
tryPickRemove(SoNode *root,
    const SbViewportRegion &viewport,
    const SbVec2s &cursorPosition)
{

    SoRayPickAction myPickAction(viewport);

    // Set an 8-pixel wide region around the pixel
    myPickAction.setPoint(cursorPosition);
    myPickAction.setRadius(8.0);

    // Start a pick traversal
    myPickAction.apply(root);
    const SoPickedPoint *myPickedPoint =
            myPickAction.getPickedPoint();
    if (myPickedPoint == NULL) return FALSE;

    SoNode* node =  myPickedPoint->getPath()->getNodeFromTail(1);
    course->remove_circle((SoSeparator *)node);

    return TRUE;
}



// Direct Event Handling
SbBool
myAppEventHandler(void *userData, XAnyEvent *anyevent)
{
    SoXtRenderArea *myRenderArea = (SoXtRenderArea *) userData;
    XButtonEvent *myButtonEvent;
    XMotionEvent *myMotionEvent;
    XKeyEvent *myKeyEvent;
    SbVec3f vec;
    SbBool handled = TRUE;

    const SbViewportRegion &myRegion =
    myRenderArea->getViewportRegion();
    SoGroup *root = (SoGroup *) myRenderArea->getSceneGraph();

    short window_w, window_h;
    myRegion.getWindowSize().getValue(window_w,window_h);

    SbVec3f picked_position;

    switch (anyevent->type) {

    case ButtonPress:
       myButtonEvent = (XButtonEvent *) anyevent;
       if (myButtonEvent->button == Button1) {
         const SbVec2s mouse_position
           (myButtonEvent->x, window_h - myButtonEvent->y);
         if (tryPick(root, myRegion, mouse_position, &picked_position)){
           //print_position(picked_position);
           float x,y,z;
           picked_position.getValue(x,y,z);
           course->add_circle(x,z,1);
         }
       } else if (myButtonEvent->button == Button2) {
```

```
        const SbVec2s mouse_position
          (myButtonEvent->x, window_h - myButtonEvent->y);
        tryPickRemove(root, myRegion, mouse_position);
      } else if (myButtonEvent->button == Button3) {
      }
      break;

   case ButtonRelease:
      myButtonEvent = (XButtonEvent *) anyevent;
      if (myButtonEvent->button == Button1) {
      }
      break;

   case MotionNotify:
      myMotionEvent = (XMotionEvent *) anyevent;
      if (myMotionEvent->state & Button1Mask) {
        const SbVec2s mouse_position
          (myMotionEvent->x, window_h - myMotionEvent->y);
        if (tryPick(root, myRegion, mouse_position, &picked_position)){
          float x,y,z;
          picked_position.getValue(x,y,z);
          course->resize_circle(x,z);
        }
      }
      break;
   case KeyPress:
      myKeyEvent = (XKeyEvent *) anyevent;
      course->save(myKeyEvent->keycode);
      break;

   default:
      handled = FALSE;
      break;
   }

   return handled;
}



SoSeparator* load(char *filename){
   // Read the object from a file and add to the scene
   SoInput myInput;
   if (!myInput.openFile(filename) )
      exit (1);
   SoSeparator *fileContents = SoDB::readAll(&myInput);
   if (fileContents == NULL)
      exit (1);
   return fileContents;
}


void save(SoSeparator *root, char *filename){
   SoWriteAction myAction;
   myAction.getOutput()->openFile(filename);
   myAction.getOutput()->setBinary(FALSE);
   myAction.apply(root);
   myAction.getOutput()->closeFile();
}

void save_binary(SoSeparator *root, char *filename){
   SoWriteAction myAction;
   myAction.getOutput()->openFile(filename);
   myAction.getOutput()->setBinary(TRUE);
   myAction.apply(root);
```



```
   myAction.getOutput()->closeFile();
}
```

```
class Jump{
  public:
    float jump_dx;
    float jump_dz;
    int   jump_step;
    float jump_dangle;
    SbRotation jump_drotation;

    float goal_height;  // for direct jump

    SoTranslation *markPosition;


Jump(SoGroup *parent){
    SoSeparator *mark = new SoSeparator;

    markPosition  = new SoTranslation;
    mark->addChild(markPosition);

    SoTranslation *myTranslation  = new SoTranslation;
    myTranslation->translation.setValue(0, -0.953, 0);
    mark->addChild(myTranslation);
    SoScale *myScale = new SoScale;
    myScale->scaleFactor.setValue(0.5,0.1,0.5);
    mark->addChild(myScale);
    SoPickStyle *myPickStyle = new SoPickStyle;
    myPickStyle->style = SoPickStyle::UNPICKABLE;
    mark->addChild(myPickStyle);
    SoMaterial *myMaterial = new SoMaterial;
    myMaterial->diffuseColor.setValue(1.0, 0.0, 0.0);   // Red
    mark->addChild(myMaterial);
    mark->addChild(new SoDirectionalLight);

    mark->addChild(new SoCylinder);

    parent->addChild(mark);

}




// change mark position and initialize animaiton
void
init_move(SoNode *root,
        SbVec3f position)
{
   // set mark at the destination
   float mark_x,mark_y,mark_z;
   markPosition->translation.getValue().getValue(mark_x,mark_y,mark_z);
   float pick_x, pick_y, pick_z;
   position.getValue(pick_x, pick_y, pick_z);
   markPosition->translation.setValue(pick_x, pick_y+1, pick_z);
   //markPosition->translation.setValue(pick_x, mark_y, pick_z);

   // set position animation parameters
```



```
   float goal_x,goal_y,goal_z;
   position.getValue(goal_x,goal_y,goal_z);

   float current_x,current_y,current_z;
   selfPosition->translation.getValue()
     .getValue(current_x,current_y,current_z);

   jump_dx = (goal_x - current_x);
   jump_dz = (goal_z - current_z);
   float length = sqrt(jump_dx * jump_dx + jump_dz * jump_dz);
   jump_step = int(length/speed)+1;

   goal_height = pick_y+1;

   // set direction animation parameters
   SbVec3f init_direction(0,0,-1);
   SbVec3f target_direction(jump_dx, 0, jump_dz);
   SbRotation targetRotation(init_direction, target_direction);
   //targetRotation.setValue(init_direction, target_direction);
   SbRotation currentRotation = selfRotation->rotation.getValue();

   SbVec3f dammy_axis;
   float current_radians, target_radians;
   targetRotation.getValue(dammy_axis, target_radians);
   currentRotation.getValue(dammy_axis, current_radians);

   float dAngle;
   (targetRotation*=(currentRotation.inverse()))
     .getValue(dammy_axis, dAngle);
   if (dAngle > 3.141592) dAngle = dAngle - 6.283184 ;

   if (Abs(dAngle / jump_step) > rotation_speed){
     jump_step = int(Abs(dAngle)/ rotation_speed) + 1;
   }


   float step_length = length/ jump_step;
   jump_dx = jump_dx*step_length/length;
   jump_dz = jump_dz*step_length/length;

   jump_dangle = dAngle / jump_step;
   jump_drotation.setValue(dammy_axis, jump_dangle);


   //selfRotation->rotation.setValue(myRotation);              // 現在の方向

};



SbBool
auto_move(SoSFVec3f *position, SoSFRotation *direction){

    // jump to target
    if (jump_step ==0)
      return FALSE;

    jump_step--;

    // position
    float previous_x,previous_y,previous_z;
    position->getValue().getValue(previous_x,previous_y,previous_z);

   if (course->blocked(previous_x + jump_dx, previous_z + jump_dz)){
     jump_step = 0;
```

```
      return FALSE;
    }

    position->setValue(previous_x + jump_dx,
                       previous_y,
                       previous_z + jump_dz);

    // direction
    *direction
      = selfRotation->rotation.getValue() * jump_drotation;

    return TRUE;
};


// Direct jump without animation
void
jump(SoSFVec3f *position, SoSFRotation *rotation){

   // position

   // check blocks
   float previous_x,previous_y,previous_z;
   position->getValue().getValue(previous_x,previous_y,previous_z);
   float next_x = previous_x;
   float next_z = previous_z;
   for (int i = 1; i <= jump_step; i++){
     if (course->blocked(previous_x + jump_dx*i, previous_z + jump_dz*i)){
         break;
     }
     else {
       next_x = previous_x + jump_dx*i;
       next_z = previous_z + jump_dz*i;
     }
   }
   position->setValue(next_x, goal_height, next_z);
   //position->setValue(next_x, previous_y, next_z);

   // set direction
   SbVec3f base_direction(0,0,-1);
   SbVec3f target_direction(jump_dx, 0, jump_dz);
   *rotation = SbRotation(base_direction, target_direction);

   jump_step = 0;


};

void
clear(){
   markPosition->translation.setValue(0, 1000, 0);
}


};
```

```
class KeyMove{
  public:
    SbBool forward_moving;
    SbBool backward_moving;
    SbBool right_turning;
    SbBool left_turning;


    KeyMove(){
      forward_moving = FALSE;
      backward_moving = FALSE;
      right_turning = FALSE;
      left_turning = FALSE;
    };

    SbBool moving();
    SbBool auto_move();
    void init_move(int keycode);
    void stop_move(int keycode);
    void move_forward(int direction);
    void turn_right(int direction);


};


SbBool
KeyMove::moving(){
  return (forward_moving || backward_moving || right_turning || left_turning);
}

SbBool
KeyMove::auto_move(){
  SbBool handled = FALSE;
  if (forward_moving){
       move_forward(1);
       handled = TRUE;
  }
  else if (backward_moving){
       move_forward(-1);
       handled = TRUE;
  }
  if (right_turning){
       turn_right(-1);
       handled = TRUE;
  }
  else if (left_turning){
       turn_right(1);
       handled = TRUE;
  }
  return handled;
}

void
KeyMove::move_forward(int direction){

   SbVec3f init_direction(0,0,-1);
   SbVec3f target_direction;
   selfRotation->rotation.getValue().multVec(init_direction, target_direction);
   float dx,dy,dz;
   target_direction.getValue(dx, dy, dz);

    // position
   float previous_x,previous_y,previous_z;
```



```
   selfPosition->translation.getValue()
      .getValue(previous_x,previous_y,previous_z);

   float x, z;
   x = previous_x + dx*speed*direction;
   z = previous_z + dz*speed*direction;
   if (course->blocked(x, z))
     return;

    selfPosition->translation.setValue(x,
                               previous_y,
                               z);

}
void
KeyMove::turn_right(int direction){
   SbRotation currentRotation = selfRotation->rotation.getValue();

   currentRotation*=SbRotation(
      SbVec3f(0,1,0), rotation_speed*direction);

    selfRotation->rotation
      = currentRotation;

}

void
KeyMove::init_move(int keycode){
  switch (keycode){
     case 107:
     case 62:
       forward_moving = TRUE;
       break;
     case 104:
     case 95:
       backward_moving = TRUE;
       break;
     case 114:
     case 80:
       right_turning = TRUE;
       break;
     case 105:
     case 78:
       left_turning = TRUE;
       break;
     default:
       break;
   }
}
void
KeyMove::stop_move(int keycode){
  switch (keycode){
     case 107:
     case 62:
       forward_moving = FALSE;
       break;
     case 104:
     case 95:
       backward_moving = FALSE;
       break;
     case 114:
     case 80:
       right_turning = FALSE;
       break;
     case 105:
     case 78:
```

```
        left_turning = FALSE;
        break;
      default:
        break;
    }
}
```

```
// Copyright (c) Takeo Igarashi


// Path Drawing project
//    module for demo video


// common parameters
const SbBool vista = FALSE;
SbBool animation = TRUE;
char* human_model_left = "kuma1.iv";
char* human_model_right = "kuma2.iv";

float speed = 1;
float rotation_speed = 0.05;
float interval = 0.02;
const float camera_distance = 18;
const float camera_height = 8;


#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoFaceSet.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoNormal.h>
#include <Inventor/nodes/SoNormalBinding.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoVertexProperty.h>




void
specify_vertex(float *array, float x, float y, float z){
  *(array+0) = x;
  *(array+1) = y;
  *(array+2) = z;
}




//  Eight polygons. The first four are triangles
//  The second four are quadrilaterals for the sides.
static float ground_vertices[64*4][3];
static int32_t ground_numvertices[64];
const int unit = 10;

SoSeparator *
makeGroundFaceSet()
{
   int x,y,xx,yy,xxx;

   for (y = 0; y < 8; y++)
     for (x = 0; x < 8; x++)
       for (yy = 0; yy < 2; yy++)
         for (xx = 0; xx < 2; xx++){
       if (yy==1) xxx = 1 - xx; else xxx = xx;
       specify_vertex(ground_vertices[(y*8+x)*4+yy*2+xx],
                      (x + xxx) * unit,  (y + yy) * unit, 0);
     }

   for (int i = 0; i < 64; i++)
     ground_numvertices[i] = 4;
```



```
   SoSeparator *ground = new SoSeparator();
   ground->ref();

   SoMaterial *myMaterial = new SoMaterial;
   myMaterial->emissiveColor.setValue(0.0, 0.2, 0.0);
   ground->addChild(myMaterial);

   // Using the new SoVertexProperty node is more efficient
   SoVertexProperty *myVertexProperty = new SoVertexProperty;
   //myVertexProperty->orderedRGBA.setValue(SbColor(.0,.8,.2).getPackedValue());
   myVertexProperty->orderedRGBA.setValue(SbColor(.0,1.0,.5).getPackedValue());
   myVertexProperty->vertex.setValues(0, 64*4, ground_vertices);
   SoFaceSet *myFaceSet = new SoFaceSet;
   myFaceSet->numVertices.setValues(0, 64, ground_numvertices);

   myFaceSet->vertexProperty.setValue(myVertexProperty);
   myFaceSet->setName("ground");
   ground->addChild(myFaceSet);

   ground->unrefNoDelete();
   return ground;

}


static float tower_vertices[5*4][3];
static int32_t tower_numvertices[5];
static float tower_height = unit /1.5;

SoSeparator *
makeTowerFaceSet()
{
   int x,y,xx,yy,xxx;

   int n = 0;

   x = 2;
   y = 2;
   // top
   specify_vertex(tower_vertices[n++], x * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, tower_height);

   // side top
   specify_vertex(tower_vertices[n++], x * unit, y * unit, 0);
   specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, 0);
   specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], x * unit, y * unit, tower_height);

   // side left
   specify_vertex(tower_vertices[n++], x * unit, y * unit, 0);
   specify_vertex(tower_vertices[n++], x * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, 0);

   // side bottom
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, 0);
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, 0);


   // side right
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, 0);
```

```
    specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, tower_height);
    specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, tower_height);
    specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, 0);


    for (int i = 0; i < 5; i++)
      tower_numvertices[i] = 4;


    SoSeparator *tower = new SoSeparator();
    tower->ref();

    // Using the new SoVertexProperty node is more efficient
    SoVertexProperty *myVertexProperty = new SoVertexProperty;
    myVertexProperty->orderedRGBA.setValue(SbColor(.6,.8,.8).getPackedValue());
    myVertexProperty->vertex.setValues(0, 5*4, tower_vertices);

    SoPickStyle* pickstyle = new SoPickStyle;
    pickstyle->style = SoPickStyle::UNPICKABLE;
    tower->addChild(pickstyle);

    SoFaceSet *myFaceSet = new SoFaceSet;
    myFaceSet->numVertices.setValues(0, 5, tower_numvertices);
    myFaceSet->vertexProperty.setValue(myVertexProperty);
    myFaceSet->setName("tower");
    tower->addChild(myFaceSet);

    tower->unrefNoDelete();
    return tower;
}


static float arch_vertices[3*4][3];
static int32_t arch_numvertices[3];
static float arch_height = unit / 2.5;

SoSeparator *
makeArchFaceSet()
{
    int x,y,xx,yy,xxx;

    int n = 0;

    x = 1;
    y = 5;

    // left
    specify_vertex(arch_vertices[n++], x * unit, y * unit, 0);
    specify_vertex(arch_vertices[n++], (x+2) * unit, y * unit, arch_height);
    specify_vertex(arch_vertices[n++], (x+2) * unit, (y+1) * unit, arch_height);
    specify_vertex(arch_vertices[n++], x * unit, (y+1) * unit, 0);

    // top
    specify_vertex(arch_vertices[n++], (x+2) * unit, y * unit, arch_height);
    specify_vertex(arch_vertices[n++], (x+4) * unit, y * unit, arch_height);
    specify_vertex(arch_vertices[n++], (x+4) * unit, (y+1) * unit, arch_height);
    specify_vertex(arch_vertices[n++], (x+2) * unit, (y+1) * unit, arch_height);

    // right
    specify_vertex(arch_vertices[n++], (x+4) * unit, y * unit, arch_height);;
    specify_vertex(arch_vertices[n++], (x+6) * unit, y * unit, 0);
    specify_vertex(arch_vertices[n++], (x+6) * unit, (y+1) * unit, 0);
    specify_vertex(arch_vertices[n++], (x+4) * unit, (y+1) * unit, arch_height);

    for (int i = 0; i < 3; i++)
      arch_numvertices[i] = 4;
```



```
    SoSeparator *arch = new SoSeparator();
    arch->ref();

    // Using the new SoVertexProperty node is more efficient
    SoVertexProperty *myVertexProperty = new SoVertexProperty;
    myVertexProperty->orderedRGBA.setValue(SbColor(.6,.6,.6).getPackedValue());
    myVertexProperty->vertex.setValues(0, 3*4, arch_vertices);
    SoFaceSet *myFaceSet = new SoFaceSet;
    myFaceSet->numVertices.setValues(0, 3, arch_numvertices);

    myFaceSet->vertexProperty.setValue(myVertexProperty);
    myFaceSet->setName("arch");
    arch->addChild(myFaceSet);

    arch->unrefNoDelete();
    return arch;
}
static float archside_vertices[32][3];
static int32_t archside_numvertices[9] = { 3,4,3, 3,4,3, 4,4,4};
static float archside_height = arch_height * 0.8;

SoSeparator *
makeArchsideFaceSet()
{
    int x,y,xx,yy,xxx;

    int n = 0;

    x = 1;
    y = 5;

    // top side
    specify_vertex(archside_vertices[n++], x * unit, y * unit, 0);
    specify_vertex(archside_vertices[n++], (x+2) * unit, y * unit, 0);
    specify_vertex(archside_vertices[n++], (x+2) * unit, y * unit, arch_height);

    specify_vertex(archside_vertices[n++], (x+2) * unit, y * unit, archside_height);
    specify_vertex(archside_vertices[n++], (x+4) * unit, y * unit, archside_height);
    specify_vertex(archside_vertices[n++], (x+4) * unit, y * unit, arch_height);
    specify_vertex(archside_vertices[n++], (x+2) * unit, y * unit, arch_height);

    specify_vertex(archside_vertices[n++], (x+4) * unit, y * unit, 0);
    specify_vertex(archside_vertices[n++], (x+6) * unit, y * unit, 0);
    specify_vertex(archside_vertices[n++], (x+4) * unit, y * unit, arch_height);


    // bottom   side
    specify_vertex(archside_vertices[n++], x * unit, (y+1) * unit, 0);
    specify_vertex(archside_vertices[n++], (x+2) * unit, (y+1) * unit, arch_height);
    specify_vertex(archside_vertices[n++], (x+2) * unit, (y+1) * unit, 0);

    specify_vertex(archside_vertices[n++], (x+2) * unit, (y+1) * unit, archside_height
);
    specify_vertex(archside_vertices[n++], (x+2) * unit, (y+1) * unit, arch_height);
    specify_vertex(archside_vertices[n++], (x+4) * unit, (y+1) * unit, arch_height);
    specify_vertex(archside_vertices[n++], (x+4) * unit, (y+1) * unit, archside_height
);

    specify_vertex(archside_vertices[n++], (x+4) * unit, (y+1) * unit, 0);
    specify_vertex(archside_vertices[n++], (x+4) * unit, (y+1) * unit, arch_height);
    specify_vertex(archside_vertices[n++], (x+6) * unit, (y+1) * unit, 0);

    // inner   side
    specify_vertex(archside_vertices[n++], (x+2) * unit, y * unit, 0);
    specify_vertex(archside_vertices[n++], (x+2) * unit, (y+1) * unit, 0);
```

```
   specify_vertex(archside_vertices[n++], (x+2) * unit, (y+1) * unit, archside_height
);
   specify_vertex(archside_vertices[n++], (x+2) * unit, y * unit, archside_height);

   specify_vertex(archside_vertices[n++], (x+2) * unit, y * unit, archside_height);
   specify_vertex(archside_vertices[n++], (x+2) * unit, (y+1) * unit, archside_height
);
   specify_vertex(archside_vertices[n++], (x+4) * unit, (y+1) * unit, archside_height
);
   specify_vertex(archside_vertices[n++], (x+4) * unit, y * unit, archside_height);

   specify_vertex(archside_vertices[n++], (x+4) * unit, y * unit, 0);
   specify_vertex(archside_vertices[n++], (x+4) * unit, y * unit, archside_height);
   specify_vertex(archside_vertices[n++], (x+4) * unit, (y+1) * unit, archside_height
);
   specify_vertex(archside_vertices[n++], (x+4) * unit, (y+1) * unit, 0);



   SoSeparator *archside = new SoSeparator();
   archside->ref();

   // Using the new SoVertexProperty node is more efficient
   SoVertexProperty *myVertexProperty = new SoVertexProperty;
   myVertexProperty->orderedRGBA.setValue(SbColor(.6,.6,.6).getPackedValue());
   myVertexProperty->vertex.setValues(0, 32, archside_vertices);

   SoPickStyle* pickstyle = new SoPickStyle;
   pickstyle->style = SoPickStyle::UNPICKABLE;
   archside->addChild(pickstyle);

   SoFaceSet *myFaceSet = new SoFaceSet;
   myFaceSet->numVertices.setValues(0, 9, archside_numvertices);
   myFaceSet->vertexProperty.setValue(myVertexProperty);
   myFaceSet->setName("archside");
   archside->addChild(myFaceSet);


   archside->unrefNoDelete();
   return archside;
}









class Course{
  public:
    SoSeparator *root;
    Course(){
      if (vista){
        root = load("../vista/saved.iv"); // simple
        return;
      }

      root = new SoSeparator;
      root->ref();


      SoPerspectiveCamera *camera = new SoPerspectiveCamera;
```



```
      camera->position=SbVec3f(0, 3.23168,0);
      camera->setName("camera");
      root->addChild(camera);


      SoSeparator* scene = new SoSeparator;

      SoTranslation* translation = new SoTranslation;
      translation->translation = SbVec3f(0,-1, 0);
      scene->addChild(translation);
      SoRotation* rotation = new SoRotation;
      rotation->rotation = SbRotation(SbVec3f(1,0,0), -1.5708);
      scene->addChild(rotation);


      scene->addChild(makeGroundFaceSet());
      scene->addChild(makeTowerFaceSet());
      scene->addChild(makeArchFaceSet());
      scene->addChild(makeArchsideFaceSet());


      root->addChild(scene);


    }

    SbBool blocked(float mx, float my){

      if (mx < 0 || mx > 8 * unit || my > 0 || my < -8 * unit)
        return TRUE;

      return FALSE;
    }

    float get_height(float mx, float my, float mh){
      float x = mx;
      float y = -my;
      if (x > unit*1 && x<unit*3 && y > unit*5 && y < unit*6)
        return  arch_height * (x - unit*1) / unit / 2;
      else if (x > unit*3 && x<unit*5 && y > unit*5 && y < unit*6){
        if (mh > 0.1) return arch_height;
        else return 0;
        //return  0; //arch_height;
      }
      else if (x > unit*5 && x<unit*7 && y > unit*5 && y < unit*6)
        return  arch_height * (unit*7 - x) / unit / 2;
      else
      return 0;
    }


SoFaceSet *previous_faceset;
int previous_faceindex;

SbVec3f
pickposition(SoRayPickAction *myPickAction, SbVec3f *position){
   const SoPickedPoint *myPickedPoint =
            myPickAction->getPickedPoint();

   if (button == Button2)
     return *position;

   SoFaceDetail *facedetail = (SoFaceDetail *) myPickedPoint->getDetail();
   int index = facedetail->getFaceIndex();
   SoFaceSet *faceset = (SoFaceSet *)myPickedPoint->getPath()->getTail();
   //printf(faceset->getName().getString());
```

```
    //printf(" / index %d\n", index);

    if (previous_faceset == NULL)
      previous_faceset = faceset;

    // walk under the arch
    if ((faceset->getName() == "arch")
        && (previous_faceset->getName() == "ground")) {
      SoPickedPointList l = myPickAction->getPickedPointList();
      SoPickedPoint *p = l[l.getLength() - 1];

      if (   (index == 1)
          || ((index == 0) && (previous_faceindex!=40) &&
              (previous_faceindex!=32) && (previous_faceindex!=48))
          || ((index == 2) && (previous_faceindex!=47) &&
              (previous_faceindex!=39) && (previous_faceindex!=55))){

          *position = p->getPoint();
          faceset = (SoFaceSet *)p->getPath()->getTail();
      }

    }
    // walk on the arch  make sure not to jump off
    else if ((previous_faceset->getName() == "arch")
             && (faceset->getName() == "ground")) {
      if ( (previous_faceindex == 1) ||
          ((previous_faceindex == 0) && (index != 40)) ||
          ((previous_faceindex == 2) && (index != 47)))
      return NULL;
    }
    // prevent sinking
    else if ((faceset->getName() == "arch") &&
             (previous_faceset->getName() == "arch") ){
      if (((index == 0) && (previous_faceindex == 1)) ||
          ((index == 1) && (previous_faceindex == 0))){
        (*position)[0] = unit*3;
        (*position)[1] = arch_height-1;
      }
      else if (((index == 1) && (previous_faceindex == 2)) ||
               ((index == 2) && (previous_faceindex == 1))){
        (*position)[0] = unit*5;
        (*position)[1] = arch_height-1;
      }
    }


    previous_faceset = faceset;
    previous_faceindex = index;


    avoid_building(position);

    return *position;
}



const float outer_radius = unit * 1.5;
const float inner_radius = unit * 0.8;

void
avoid_building(SbVec3f *position){

  SbVec2f original((*position)[0], (*position)[2]);
  SbVec2f center(unit*2.5, -unit*2.5);
  SbVec2f vector = original-center;
```



```
  float length = vector.length();

  if (length > outer_radius) return;

  float ratio = length / outer_radius;
  ratio =
    (inner_radius + ratio * (outer_radius - inner_radius)) / length;

    (*position)[0] = center[0] + ratio * vector[0];
    (*position)[2] = center[1] + ratio * vector[1];

  return;

}



/*
void
avoid_building(SbVec3f *position){

  float x = (*position)[0];
  float y = (*position)[2];

  if ((x < unit*2) || (x > unit*3) || (y > -unit*2) || (y < -unit*3))
    return;

  SbVec2f center(unit*2.5, -unit*2.5);
  SbVec2f original(x,y);
  SbVec2f vector = original-center;

  SbVec2f modified;
  if (vector[0]<vector[1] && -vector[0]<vector[1])
    modified = y_slice(-unit*1.98, center, vector);
  else if (vector[0]>vector[1] && -vector[0]>vector[1])
    modified = y_slice(-unit*3.02, center, vector);
  else if (vector[0]<vector[1] && vector[0]<-vector[1])
    modified  = x_slice(unit*1.98, center, vector);
  else
    modified  = x_slice(unit*3.02, center, vector);

    (*position)[0] = modified[0];
    (*position)[2] = modified[1];

  return;

}

SbVec2f
y_slice(float y, SbVec2f center, SbVec2f vector){
  float ratio = (y-center[1])/vector[1];
  float x = center[0] + vector[0] * ratio;
  //float y = center[1] + vector[1] * ratio;
return   SbVec2f(x,y);
}
SbVec2f
x_slice(float x, SbVec2f center, SbVec2f vector){
  float ratio = (x-center[0])/vector[0];
  //float x = center[0] + vector[0] * ratio;
  float y = center[1] + vector[1] * ratio;
return   SbVec2f(x,y);
}

*/
```

```
void
reset_parameters(){
   camera->position.setValue(50, camera_height, +camera_distance);

   selfPosition->translation = SbVec3f(50,0,0);
   selfRotation->rotation = SbRotation(SbVec3f(0,1,0), 0);

}
void
set_next_parameters(){
    //reset_parameters();

    animation = !animation;

    if (animation)
      interval = 0.1;
    else
      interval = 0.4;
   jumpSensor->setInterval(interval);
}
void
init_parameters(){
   // start position

   experiment_mode = KEYMOVE;
   reset_parameters();
}



);
```

```
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoFaceSet.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoNormal.h>
#include <Inventor/nodes/SoNormalBinding.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoVertexProperty.h>

void
specify_vertex(float *array, float x, float y, float z){
  *(array+0) = x;
  *(array+1) = y;
  *(array+2) = z;
}




//  Eight polygons. The first four are triangles
//  The second four are quadrilaterals for the sides.
static float ground_vertices[64*4][3];
static int32_t ground_numvertices[64];
const int unit = 8;

SoSeparator *
makeGroundFaceSet()
{
   int x,y,xx,yy,xxx;

   for (y = 0; y < 8; y++)
     for (x = 0; x < 8; x++)
       for (yy = 0; yy < 2; yy++)
         for (xx = 0; xx < 2; xx++){
       if (yy==1) xxx = 1 - xx; else xxx = xx;
       specify_vertex(ground_vertices[(y*8+x)*4+yy*2+xx],
                      (x + xxx) * unit,  (y + yy) * unit, 0);
     }

   for (int i = 0; i < 64; i++)
     ground_numvertices[i] = 4;

   SoSeparator *ground = new SoSeparator();
   ground->ref();

   // Using the new SoVertexProperty node is more efficient
   SoVertexProperty *myVertexProperty = new SoVertexProperty;
   myVertexProperty->orderedRGBA.setValue(SbColor(.0,.8,.2).getPackedValue());
   myVertexProperty->vertex.setValues(0, 64*4, ground_vertices);
   SoFaceSet *myFaceSet = new SoFaceSet;
   myFaceSet->numVertices.setValues(0, 64, ground_numvertices);

   myFaceSet->vertexProperty.setValue(myVertexProperty);
   myFaceSet->setName("ground");
   ground->addChild(myFaceSet);

   ground->unrefNoDelete();
   return ground;

}


static float tower_vertices[5*4][3];
```



```
static int32_t tower_numvertices[5];
static float tower_height = unit *2;

SoSeparator *
makeTowerFaceSet()
{
   int x,y,xx,yy,xxx;

   int n = 0;

   x = 2;
   y = 2;
   // top
   specify_vertex(tower_vertices[n++], x * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, tower_height);

   // side top
   specify_vertex(tower_vertices[n++], x * unit, y * unit, 0);
   specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, 0);
   specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], x * unit, y * unit, tower_height);

   // side left
   specify_vertex(tower_vertices[n++], x * unit, y * unit, 0);
   specify_vertex(tower_vertices[n++], x * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, 0);

   // side bottom
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, 0);
   specify_vertex(tower_vertices[n++], x * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, 0);


   // side right
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, 0);
   specify_vertex(tower_vertices[n++], (x+1) * unit, (y+1) * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, tower_height);
   specify_vertex(tower_vertices[n++], (x+1) * unit, y * unit, 0);


   for (int i = 0; i < 5; i++)
     tower_numvertices[i] = 4;


   SoSeparator *tower = new SoSeparator();
   tower->ref();

   // Using the new SoVertexProperty node is more efficient
   SoVertexProperty *myVertexProperty = new SoVertexProperty;
   myVertexProperty->orderedRGBA.setValue(SbColor(.6,.6,.6).getPackedValue());
   myVertexProperty->vertex.setValues(0, 5*4, tower_vertices);
   SoFaceSet *myFaceSet = new SoFaceSet;
   myFaceSet->numVertices.setValues(0, 5, tower_numvertices);

   myFaceSet->vertexProperty.setValue(myVertexProperty);
   myFaceSet->setName("tower");
   tower->addChild(myFaceSet);

   tower->unrefNoDelete();
   return tower;
}
```

```
static float arch_vertices[3*4][3];
static int32_t arch_numvertices[3];
static float arch_height = unit / 2.0;

SoSeparator *
makeArchFaceSet()
{
   int x,y,xx,yy,xxx;

   int n = 0;

   x = 2;
   y = 5;

   // left
   specify_vertex(arch_vertices[n++], x * unit, y * unit, 0);
   specify_vertex(arch_vertices[n++], (x+1) * unit, y * unit, arch_height);
   specify_vertex(arch_vertices[n++], (x+1) * unit, (y+1) * unit, arch_height);
   specify_vertex(arch_vertices[n++], x * unit, (y+1) * unit, 0);

   // top
   specify_vertex(arch_vertices[n++], (x+1) * unit, y * unit, arch_height);
   specify_vertex(arch_vertices[n++], (x+3) * unit, y * unit, arch_height);
   specify_vertex(arch_vertices[n++], (x+3) * unit, (y+1) * unit, arch_height);
   specify_vertex(arch_vertices[n++], (x+1) * unit, (y+1) * unit, arch_height);

   // right
   specify_vertex(arch_vertices[n++], (x+3) * unit, y * unit, arch_height);;
   specify_vertex(arch_vertices[n++], (x+4) * unit, y * unit, 0);
   specify_vertex(arch_vertices[n++], (x+4) * unit, (y+1) * unit, 0);
   specify_vertex(arch_vertices[n++], (x+3) * unit, (y+1) * unit, arch_height);

   for (int i = 0; i < 3; i++)
     arch_numvertices[i] = 4;

   SoSeparator *arch = new SoSeparator();
   arch->ref();

   // Using the new SoVertexProperty node is more efficient
   SoVertexProperty *myVertexProperty = new SoVertexProperty;
   myVertexProperty->orderedRGBA.setValue(SbColor(.6,.6,.6).getPackedValue());
   myVertexProperty->vertex.setValues(0, 3*4, arch_vertices);
   SoFaceSet *myFaceSet = new SoFaceSet;
   myFaceSet->numVertices.setValues(0, 3, arch_numvertices);

   myFaceSet->vertexProperty.setValue(myVertexProperty);
   myFaceSet->setName("arch");
   arch->addChild(myFaceSet);

   arch->unrefNoDelete();
   return arch;
}

void
main(int, char **argv)
{
   // Initialize Inventor and Xt
   Widget myWindow = SoXt::init(argv[0]);
   if (myWindow == NULL) exit(1);

   SoSeparator *root = new SoSeparator;
   root->ref();

   root->addChild(makeGroundFaceSet());
   root->addChild(makeTowerFaceSet());
```



```
   root->addChild(makeArchFaceSet());

   SoXtExaminerViewer *myViewer =
           new SoXtExaminerViewer(myWindow);
   myViewer->setSceneGraph(root);
   myViewer->setTitle("Face Set: Ground");
   myViewer->show();
   myViewer->viewAll();

   SoXt::show(myWindow);
   SoXt::mainLoop();
}
```

```
#include <Inventor/nodes/SoTriangleStripSet.h>

// wire line
static float vertices[5][3] =
{
  {0,0,0}, {0,1,0}, {1,1,0}, {1,0,0}, {0,0,0}
};
static int32_t numvertices[1] = {0};


// Strip
static float vertexPositions[4][3] =
{
  {50,0,0},{51,0,0},{50,0,-10},{51,0,-10}
};
static int32_t numVertices[1]=
{
   0 // flag
};



class Path {
 public:
  SoGroup *parent;
  SoSeparator *line;
  SoCoordinate3 *myCoords;
  SoLineSet *myLineSet;

  float ax, bx, cx, dx,
        ay, by, cy, dy;
  float distance;
  float t;
  int current_index;
  SbBool turning;

  SbRotation camera_dRotation;

  Path(SoGroup *r){
    parent = r;

    line = new SoSeparator;

    // Define coordinates for vertices
    myCoords = new SoCoordinate3;
    myCoords->point.setValues(0, 0, vertices);
    line->addChild(myCoords);

    SoPickStyle *myPickStyle = new SoPickStyle;
    myPickStyle->style = SoPickStyle::UNPICKABLE;
    line->addChild(myPickStyle);

    SoMaterial *myMaterial = new SoMaterial;
    myMaterial->diffuseColor.setValue(1.0, 0.0, 0.0);   // Red
    line->addChild(myMaterial);
    line->addChild(new SoDirectionalLight);

    myLineSet = new SoLineSet;
    myLineSet->numVertices.setValues(0, 1, numvertices);
    line->addChild(myLineSet);

    parent->addChild(line);


    createStrip(parent);
```



```
  }

SoTriangleStripSet *strips;
SoVertexProperty*  strips_vertexProperty;
const float path_width=0.4;

void
createStrip(SoGroup *parent){
   SoSeparator *result = new SoSeparator;
   result->ref();

   SoPickStyle* pickstyle = new SoPickStyle;
   pickstyle->style = SoPickStyle::UNPICKABLE;
   result->addChild(pickstyle);

   SoMaterial *myMaterial = new SoMaterial;
   myMaterial->emissiveColor.setValue(1.0, 0.0, 0.0);   // Red
   result->addChild(myMaterial);

   SoVertexProperty *myVertexProperty = new SoVertexProperty;
   myVertexProperty->orderedRGBA
       .set1Value(0, SbColor(1,0,0).getPackedValue());
   myVertexProperty->vertex.setValues(0, 4, vertexPositions);

   SoTriangleStripSet *myStrips = new SoTriangleStripSet;
   myStrips->numVertices.setValues(0, 1, numVertices);
   myStrips->vertexProperty.setValue(myVertexProperty);
   result->addChild(myStrips);

   strips = myStrips;
   strips_vertexProperty = myVertexProperty;

   parent->addChild(result);
}




void
addPoint(SbVec3f point)
{
   float x,y,z;
   point.getValue(x,y,z);
   y = y + 0.1;
   point.setValue(x,y,z);

   myCoords->point.set1Value(myCoords->point.getNum(), point);
   numvertices[0] = myCoords->point.getNum();
   myLineSet->numVertices.setValues(0, 1, numvertices);

   if (myCoords->point.getNum() > 1){
     int last = myCoords->point.getNum()-1;
     float x0 = myCoords->point[last-1][0];
     float x1 = myCoords->point[last][0];
     float y0 = myCoords->point[last-1][2];
     float y1 = myCoords->point[last][2];
     float h0 = myCoords->point[last-1][1];
     float h1 = myCoords->point[last][1];
     float vx = x1-x0;
     float vy = y1-y0;
     float length = sqrt(vx*vx + vy*vy);
     float dx = -vy/length * path_width;
     float dy = vx/length * path_width;
```

```
    int n = 4*(last-1);
    strips_vertexProperty->vertex.set1Value(n++, x0+dx, h0, y0+dy);
    strips_vertexProperty->vertex.set1Value(n++, x0-dx, h0, y0-dy);
    strips_vertexProperty->vertex.set1Value(n++, x1+dx, h1, y1+dy);
    strips_vertexProperty->vertex.set1Value(n++, x1-dx, h1, y1-dy);

    numVertices[0] =  strips_vertexProperty->vertex.getNum();
    strips->numVertices.setValues(0, 1, numVertices);
  }


}

void
clear()
{
  myCoords->point.deleteValues(0); //, myCoords->point.getNum());
  numvertices[0] = 0;
  myLineSet->numVertices.setValues(0, 1, numvertices);

  strips_vertexProperty->vertex.deleteValues(0);
  numVertices[0] = 0;
  strips->numVertices.setValues(0, 1, numVertices);
}


float
mark_x(int index){
return myCoords->point[index][0];
}
float
mark_y(int index){
return myCoords->point[index][2];
}
float
mark_h(int index){
return myCoords->point[index][1];
}
int
marks_num(){
return myCoords->point.getNum() - 1;
}




// original bezier definition
void
set_bezier(float vx1, float vx2, float x1, float x2,
                  float vy1, float vy2, float y1, float y2){

  ax = -2.0 * x2 + 2.0 * x1 + vx2 + vx1;
  bx =  3.0 * x2 - 3.0 * x1 - vx2 - 2.0 * vx1;
  cx =  vx1;
  dx =  x1;

  ay = -2.0 * y2 + 2.0 * y1 + vy2 + vy1;
  by =  3.0 * y2 - 3.0 * y1 - vy2 - 2.0 * vy1;
  cy =  vy1;
  dy =  y1;
}
// bezier with 3 points  (current direction + current position + target1,2)
void
```



```
set_bezier3(float vx, float vy, float x1, float x2,float x3,
                        float y1, float y2,float y3){
  float vx1,vy1, vx2, vy2, d2;

  distance = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)) ;
  vx1 = vx * distance;
  vy1 = vy * distance;
  //vx1 = cos(d * 3.14159 / 180.0) * distance;
  //vy1 = sin(d * 3.14159 / 180.0) * distance;

  d2 = sqrt((x3-x1)*(x3-x1)+(y3-y1)*(y3-y1)) ;
  vx2 = (x3 - x1) * distance / d2;
  vy2 = (y3 - y1) * distance / d2;

  set_bezier(vx1, vx2, x1,x2, vy1,vy2,y1,y2);
}


// bezier with 2 points (current direction + current position + target)
void
set_bezier2(float vx, float vy, float x1, float x2,
                        float y1, float y2){

  float  vx1,vy1,vx2,vy2, d2, d3;

  distance = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)) ;
  vx1 = vx * distance;
  vy1 = vy * distance;
  //vx1 = cos(d * 3.14159 / 180.0) * distance;
  //vy1 = sin(d * 3.14159 / 180.0) * distance;

  vx2 = x2 - x1;
  vy2 = y2 -y1;

  set_bezier(vx1, vx2, x1,x2, vy1,vy2,y1,y2);
}



float
bezier_x(float t){
  return ax * t * t * t + bx * t * t + cx * t + dx;
}
float
bezier_y(float t){
  return ay * t * t * t + by * t * t + cy * t + dy;
}

float
bezier_vx(float t){
  return 3.0 * ax *  t * t + 2.0 * bx * t + cx ;
}
float
bezier_vy(float t){
  return 3.0 * ay *  t * t + 2.0 * by * t + cy ;
}

float
get_direction(float x, float y){
    float direction;

    direction = atan(y/x) /3.141592 * 180;
    if (x<0)  direction = direction + 180;

    return direction;
}
```

```
void
init_vezier(SoSFVec3f *position, SoSFRotation *rotation,
                  float remaining){

    float distance;
    while(remaining >0) {
      if (current_index == marks_num()){  // marks_num()-1
        current_index--;
        t = 1;
        break;
      }
      distance = get_distance(
         mark_x(current_index+1),mark_y(current_index+1),
         mark_x(current_index),mark_y(current_index));
      t = remaining / distance;
      if (remaining < distance) break;
      remaining = remaining - distance;
      current_index++;
    }


    //if (current_index >= marks_num()) {
    //  return;
    //}

    float current_x,current_y,current_z;
    float myself_x, myself_y;
    position->getValue()
      .getValue(current_x,current_y,current_z);
    myself_x = current_x;
    myself_y = current_z;

   /* パス計算 */
     SbVec3f src(0,0,-1);
     SbVec3f dest;
     rotation->getValue().multVec(src, dest);
     float vx,vy,dammy;
     dest.getValue(vx, dammy, vy);

     if (marks_num() - current_index >=2)
       set_bezier3(vx, vy, //myself.direction,
                   myself_x,
                   mark_x(current_index), mark_x(current_index+1),
                   myself_y,
                   mark_y(current_index), mark_y(current_index+1));
     else //  (marks_num() - current_index ==1)
       set_bezier2(vx, vy, //myself.direction,
                   myself_x, mark_x(current_index),
                   myself_y, mark_y(current_index));
}

// TRUE moved FALSE not moved
SbBool
auto_move(SoSFVec3f *position, SoSFRotation *rotation){

    if (current_index >= marks_num()) {
      return FALSE;
    }

    float current_x,current_y,current_z;
    float myself_x, myself_y, myself_h;
    position->getValue()
      .getValue(current_x,current_y,current_z);
```



```
    myself_x = bezier_x( t );
    myself_y = bezier_y( t );


    if (course->blocked(myself_x, myself_y)){
      current_index = marks_num();
      return FALSE;
    }


    float prev_h = mark_h(current_index);
    float next_h = mark_h(current_index+1);
    myself_h = prev_h * (1-t) + next_h * t -1; //+1

    // ground ---- myself_h = 0.99794
    // if ( (myself_h > 0.1) &&
    float floor_level = course->get_height(myself_x, myself_y, myself_h) ;
    if (myself_h < floor_level)
       myself_h = floor_level;

    position->setValue(myself_x, myself_h, myself_y);

    SbVec3f base_direction(0,0,-1);
    SbVec3f target_direction(bezier_vx(t), 0, bezier_vy(t));
    *rotation = SbRotation(base_direction, target_direction);
    //SbRotation target_rotation(base_direction, target_direction);
    // *rotation = check_overRotation(rotation, target_rotation);

    if (t ==1 && current_index == marks_num()-1) {
       current_index = marks_num();
       return TRUE;
    }

    if (!turning)
      t = t + speed/distance;

    // 到着
    if (t > 1) {
      float remaining = (t - 1) * distance;
      current_index++;
      init_vezier(position, rotation, remaining);
    }

    return TRUE;

}

float
path_length(){
  float length = 0;
  for (int i = 0; i < marks_num()-1; i++)
    length += get_distance(mark_x(i), mark_y(i),
                           mark_x(i+1), mark_y(i+1));
  return length;
}

SbVec3f
target_position(){
return SbVec3f(mark_x(marks_num()-1),-1, mark_y(marks_num()-1));
}

SbBool
init_move(SoSFVec3f *position, SoSFRotation *rotation){

  if (marks_num() < 3) return FALSE; // jump!
```

```
  current_index = 1;
  t = 0;
  turning = FALSE;
  init_vezier(position, rotation, 0);

  //printf("n:  %d\n", marks_num());

  for(int i = 0 ; i < marks_num()-1; i++)
    draw_length += get_distance(mark_x(i), mark_y(i),
                                mark_x(i+1),mark_y(i+1));

  return TRUE;


  /*
  // camera rotation (  proportional rotation )
  SbVec3f init_direction(0,0,-1);
  float dx = mark_x(marks_num()-1) - mark_x(marks_num()-2);
  float dy = mark_y(marks_num()-1) - mark_y(marks_num()-2);;
  SbVec3f target_direction(dx, 0, dy);
  SbRotation targetRotation(init_direction, target_direction);
  SbRotation currentRotation = cameraRotation->getValue();

  SbVec3f dammy_axis;
  float current_radians, target_radians;
  targetRotation.getValue(dammy_axis, target_radians);
  currentRotation.getValue(dammy_axis, current_radians);

  float camera_dangle;
  (targetRotation*=(currentRotation.inverse()))
     .getValue(dammy_axis, camera_dangle);
  if (camera_dangle > 3.141592) camera_dangle = camera_dangle - 6.283184 ;

  int step = path_length()/speed;
  camera_dangle = camera_dangle / step;

  camera_dRotation.setValue(dammy_axis, camera_dangle);
  */
}

SbRotation
check_overRotation(SoSFRotation *rotation,
                   SbRotation targetRotation){
  SbRotation currentRotation = rotation->getValue();

  SbVec3f dammy_axis;
  float current_radians, target_radians;
  targetRotation.getValue(dammy_axis, target_radians);
  currentRotation.getValue(dammy_axis, current_radians);

  float dAngle;
  SbRotation dRotation = targetRotation * (currentRotation.inverse());
  dRotation.getValue(dammy_axis, dAngle);
  if (dAngle > 3.141592) dAngle = dAngle - 6.283184 ;

  if (Abs(dAngle) < rotation_speed){
    turning = FALSE;
    return targetRotation;
  }
  else{
    turning = TRUE;
    return currentRotation
      * SbRotation(dammy_axis, Sign(dAngle)*rotation_speed);
  }
```



```
}


// Direct jump without animation
void
jump(SoSFVec3f *position, SoSFRotation *rotation){

    if (marks_num() < 3) return;

  for(int j = 0 ; j < marks_num()-1; j++)
    draw_length += get_distance(mark_x(j), mark_y(j),
                                mark_x(j+1),mark_y(j+1));


    float x,current_y, z;
    position->getValue()
      .getValue(x, current_y, z);

    SbBool failed = FALSE;

    SbRotation direction;
    for (int i = 2; i < marks_num(); i++)
      if (!jump_sub(i, &x, &z, &direction)){
        failed = TRUE;
        break;
      }

    if (failed){
      position->setValue(x, current_y, z);
      *rotation = direction;
    }
    else{
      position->setValue(mark_x(marks_num()-1),
                         mark_h(marks_num()-1)+1, //current_y,
                         mark_y(marks_num()-1));
      *rotation = direction;
    }
}

// check if blocked along the path
SbBool
jump_sub(int n, float *x, float *y, SbRotation *direction){

  float next_x = mark_x(n);
  float next_y = mark_y(n);
  float prev_x = *x;
  float prev_y = *y;

  float dx = next_x - prev_x;
  float dy = next_y - prev_y;
  float length = sqrt(dx*dx + dy*dy);
  int step = int(length / speed + 1);


  // set direction
  SbVec3f base_direction(0,0,-1);
  SbVec3f target_direction(dx, 0, dy);
  *direction = SbRotation(base_direction, target_direction);

  dx = dx / step;
  dy = dy / step;

  *x = prev_x;
  *y = prev_y;
  for (int i = 1; i <= step; i++){
```

```
      if (course->blocked(prev_x + dx*i, prev_y + dy*i)){
          return FALSE;
      }
      else {
        *x = prev_x + dx*i;
        *y = prev_y + dy*i;
      }
  }
  return TRUE;

}


};
```

```
float Abs(float x){
  if (x>0)
    return x;
  else
    return -x;
}
float Sign(float x){
  if (x>0)
    return 1;
  else
    return -1;
}
float
get_distance(float x0, float y0, float x1, float y1){
  return sqrt((x1-x0)*(x1-x0)+(y1-y0)*(y1-y0));
}
int
mod(int original, int diameter){
  int base = (int) (original / diameter);
  return original - base * diameter;
}
```